

"十二五"普通高等教育本科国家级规划教材



21世纪大学本科计算机专业系列教材

张晨曦 骆焦煌 刘依 沈立 编著

计算机系统结构学习指导与题解（第2版）

http://www.tup.com.cn

- 根据教育部"高等学校计算机科学与技术专业规范"组织编写
- 与美国 ACM 和 IEEE *Computing Curricula 2005* 同步

清华大学出版社

普通高等教育“十二五”国家级规划教材
21 世纪大学本科计算机专业系列教材

计算机系统结构学习指导与题解

（第 2 版）

张晨曦　骆焦煌　刘　依　沈　立　编著

清华大学出版社
北　京

内容简介

本书是普通高等教育"十二五"国家级规划教材《计算机系统结构》(套书)中的一册,是《计算机系统结构教程(第2版)》(清华大学出版社)的配套教材。全书共分为14章,内容覆盖面广,包括计算机系统结构的基础知识、指令系统的设计、流水线技术、向量处理机、指令级并行性及其开发——硬件方法、指令级并行的开发——软件方法、存储系统、输入输出系统、互连网络、多处理机、多核架构与编程、机群系统、阵列处理机、数据流计算机。每一章都由4节组成,分别是基本要求与难点、知识要点、习题以及题解。知识要点给出了各章的精华和要点。习题的类型有概念题、选择题、填空题、简答题和应用题。对于应用题,书中给出了详细的求解过程。

本书概念清晰,重点难点突出,覆盖面广,题型多样,是一本很有用的学习辅导书。本书可作为计算机系统结构课程(上课或自学)的学习辅导书,也可作为计算机专业硕士研究生入学考试的复习指导书。

图书在版编目(CIP)数据

计算机系统结构学习指导与题解/张晨曦,骆焦煌等编著.—2版.—北京:清华大学出版社,2018
(21世纪大学本科计算机专业系列教材)
ISBN 978-7-302-49605-2

Ⅰ. ①计… Ⅱ. ①张… ②骆… Ⅲ. ①计算机体系结构—高等学校—教学参考资料 Ⅳ. ①TP303

中国版本图书馆CIP数据核字(2018)第028846号

责任编辑: 魏江江 薛 阳
封面设计: 何凤霞
责任校对: 梁 毅
责任印制: 丛怀宇

出版发行: 清华大学出版社
网 址: http://www.tup.com.cn, http://www.wqbook.com
地 址: 北京清华大学学研大厦A座 **邮 编:** 100084
社 总 机: 010-62770175 **邮 购:** 010-62786544
投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn
课件下载: http://www.tup.com.cn,010-62795954
印 装 者: 三河市君旺印务有限公司
经 销: 全国新华书店
开 本: 185mm×260mm **印 张:** 19 **字 数:** 464千字
版 次: 2009年10月第1版 2018年8月第2版 **印 次:** 2018年8月第1次印刷
定 价: 49.00元

产品编号:057275-01

前　言

本书是普通高等教育“十二五”国家级规划教材《计算机系统结构》(套书)中的一册，是《计算机系统结构教程(第2版)》(清华大学出版社)的配套教材。

计算机系统结构是计算机专业及相关专业的一门重要的专业课程。本书是专门为该课程编写的辅导书，既可作为该课程的教学参考书，也可作为自学该课程的辅导书，还可作为计算机专业硕士研究生入学考试的复习指导书。

全书共有14章，各章内容如下。

第1章讲述计算机系统结构的概念以及系统结构和并行性的发展，并介绍定量分析基础。

第2章是指令系统的设计，讲述计算机指令系统设计的各个方面。

第3章是流水线技术，讲述流水线的基本概念、分类和性能分析、非线性流水线的调度，介绍流水线中的相关和冲突问题及其解决方法，并讨论流水线的实现。

第4章是向量处理机，讲述向量处理机的结构、特点和性能评价。

第5章和第6章分别讲述如何用硬件和软件的方法来开发指令级并行。

第7章是存储系统，讲述Cache的基本知识以及提高Cache性能的方法，并对并行主存和虚拟存储器做了简要的讨论。

第8章是输入输出系统，讲述总线及其与CPU/存储器的连接、通道处理机及其流量分析，并详细论述了廉价磁盘冗余阵列RAID。

第9章是互连网络，讲述互连函数、互连网络的特性参数、静态互连网络、动态互连网络以及消息传递机制等。

第10章是多处理机，讲述对称式共享存储器系统结构、分布式共享存储器系统结构、多Cache一致性、同步、同时多线程以及MPP。

第11章是多核架构与编程，讲述Intel x86和ARM11 MPCore多核架构以及基于多核的并行程序设计。

第12章是机群系统，讲述机群的结构、软件模型以及机群的分类。

第13章是阵列处理机，讲述阵列处理机的操作模型、结构和特点以及并行算法。

第14章是数据流计算机，讲述数据流计算机模型、数据流程序图和数据流语言、静态数据流计算机结构以及动态数据流计算机结构。

每一章都由4节组成，分别是基本要求与难点、知识要点、习题以及题解。知识要点给出了各章的精华和要点。习题的类型有概念题、选择题、填空题、简答题和应用题。对于应用题，书中给出了详细的求解过程。

本书概念清晰，重点难点突出，覆盖面广，题型多样，是一本很有用的学习辅导书。

由于作者水平有限，书中难免有疏漏和不妥之处，敬请读者批评指正。

作　者

2017年7月

目　录

第 1 章　计算机系统结构的基础知识

1.1　基本要求与难点

1.1.1　基本要求

(1) 掌握计算机系统结构的基本概念，如：计算机系统的多级层次结构，虚拟机，计算机系统结构的定义，透明性，系列机等。

(2) 理解计算机系统的多级层次结构。

(3) 掌握计算机系统结构、计算机组成和计算机实现的定义，了解其关系。

(4) 了解计算机系统结构的分类方法。

(5) 掌握计算机系统设计的定量原理，能熟练地应用 Amdahl 定律和 CPU 性能公式进行定量分析。

(6) 理解进行计算机系统设计的 3 种主要方法，重点掌握"由中间开始"的设计方法。

(7) 理解进行计算机系统性能评测的基准测试程序法，了解如何进行计算机系统性能的比较。

(8) 理解冯·诺依曼结构的特点及其改进。

(9) 掌握实现软件移植的 3 种方法，理解系列机的概念及其根本特征。

(10) 理解并行性的定义以及并行性的不同等级，掌握提高并行性的 3 种技术途径。

(11) 了解单机系统中和多机系统中并行性的发展。

1.1.2　难点

(1) 计算机系统结构、计算机组成和计算机实现三者的定义与关系。如何判断某种技术是属于哪个方面的？如何判断其透明性？

(2) Amdahl 定律和 CPU 性能公式，如何应用它们进行定量分析？

1.2　知识要点

1.2.1　计算机系统结构的基本概念

1. 计算机系统的层次结构

可以把计算机系统按功能划分成**多级层次结构**：微程序机器级，传统机器语言机器级，

操作系统机器级,汇编语言机器级,高级语言机器级,应用语言机器级等。在这6级层次中,最下面的两级机器是用硬件/固件实现的,称为**物理机**;而上面4层一般是由软件实现的,称为**虚拟机**。

各机器级的实现主要靠翻译或解释,或两者的结合。一般来说,上述6级层次中的下面3级是用解释实现,而上面3级则是用翻译的方法实现。**翻译**是先用转换程序把高一级机器上的程序转换为低一级机器上等效的程序,然后再在这低一级机器上运行,实现程序的功能。**解释**则是对于高一级机器上的程序中的每一条语句或指令,都是转去执行低一级机器上的一段等效程序,执行完后,再去高一级机器取下一条语句或指令,再进行解释执行,如此反复,直到解释执行完整个程序。

2. 计算机系统结构的定义

计算机系统结构(Computer Architecture)的经典定义是1964年Amdahl在介绍IBM360系统时提出的:计算机系统结构是指传统机器程序员所看到的计算机属性,即概念性结构与功能特性。计算机系统结构的实质是确定计算机系统中软硬件的界面,界面之上是软件实现的功能,界面之下是硬件和固件实现的功能。

传统机器级所存在的差别对于高级语言程序员来讲是"看不见"的。在计算机技术中,把这种本来存在的事物或属性,但从某种角度看又好像不存在的概念称为**透明性**。

在J. L. Hennessy和D. A. Patterson编写的*Computer Architecture: A Quantitative Approach*一书中,把系统结构定义为囊括计算机设计的3个方面:指令系统结构,组成,硬件。我们不妨将之理解为广义的系统结构定义。

3. 计算机组成和计算机实现

计算机组成指的是计算机系统结构的逻辑实现,包含物理机器级中的数据流和控制流的组成以及逻辑设计等。它着眼于物理机器级内各事件的排序方式与控制方式、各部件的功能以及各部件之间的联系。

计算机实现指的是计算机组成的物理实现,包括处理机、主存等部件的物理结构,器件的集成度和速度,模块、插件、底板的划分与连接,信号传输,电源、冷却及整机装配技术等。它着眼于器件技术和微组装技术,其中,器件技术在实现技术中起主导作用。

具有相同系统结构的计算机因为速度、价格等方面要求的不同,可以采用不同的计算机组成。而同一种计算机组成又可以采用多种不同的计算机实现。系列机的出现充分反映了这一点。系列机是指由同一厂家生产的具有相同系统结构但具有不同组成和实现的一系列不同型号的计算机。

4. 计算机系统结构的分类

1) Flynn分类法

Flynn分类法是按照指令流和数据流的多倍性进行分类。Flynn分类法中定义:

指令流:计算机执行的指令序列。

数据流:由指令流调用的数据序列。

多倍性:在系统最受限的部件上,同时处于同一执行阶段的指令或数据的最大数目。

Flynn分类法把计算机系统的结构分为以下4类。

(1) 单指令流单数据流(Single Instruction stream Single Data stream,SISD);

(2) 单指令流多数据流(Single Instruction stream Multiple Data stream,SIMD);

(3) 多指令流单数据流(Multiple Instruction stream Single Data stream,MISD);

(4) 多指令流多数据流(Multiple Instruction stream Multiple Data stream,MIMD)。

SISD 是传统的顺序处理计算机;SIMD 以阵列处理机为代表;MISD 只是一种人为的划分,目前没有实际的机器;多处理机属于 MIMD 结构。

2) 冯氏分类法

冯氏分类法是用系统的最大并行度对计算机进行分类。最大并行度 P_m 定义为:计算机系统在单位时间内能够处理的最大的二进制位数。

按照这种分类法,可以把计算机分成以下 4 类具有不同最大并行度的计算机系统结构:字串位串,字串位并,字并位串,字并位并。

3) Handler 分类法

这种分类方法把计算机的硬件结构分成 3 个层次,并考虑它们的可并行-流水处理程度。

$$t(\text{系统型号}) = (k \times k', d \times d', w \times w')$$

这 3 个层次是:

(1) 程序控制部件(PCU)的个数 k;

(2) 算术逻辑部件(ALU)或处理部件(PE)的个数 d;

(3) 每个算术逻辑部件包含基本逻辑线路(ELC)的套数 w。

1.2.2　计算机系统的设计

1. 计算机系统设计的定量原理

1) 以经常性事件为重点

这是计算机设计中最重要、最广泛采用的设计原则。它是指要对经常发生的情况采用优化方法的原则,因为这样能得到更多的总体上的改进。这里的优化是指分配更多的资源、达到更高的性能等。

2) Amdahl 定律

Amdahl 定律告诉我们:当对一个系统中的某个部件进行改进后,所能获得的整个系统性能的提高,受限于该部件的执行时间占总执行时间的百分比。它可以用来具体地计算:当对计算机系统中的某个部分进行改进后,所能获得的加速比的大小。

这个加速比的大小与两个因素有关。一个是可改进比例,记为 Fe。另一个是可改进部件改进以后性能提高的倍数,简称部件加速比,记为 Se。改进后程序的总执行时间为:

$$T_n = T_0\left(1 - \mathrm{Fe} + \frac{\mathrm{Fe}}{\mathrm{Se}}\right)$$

其中,T_0 为改进前整个程序的执行时间,1－Fe 为不可改进比例。

改进后,整个系统的加速比为:

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1 - \mathrm{Fe}) + \dfrac{\mathrm{Fe}}{\mathrm{Se}}}$$

当 Se→∞时，则 S_n → 1/(1－Fe)。即如果只针对整个任务的一部分进行改进和优化，那么所获得的加速比不超过 1/(1－Fe)。

3) CPU 性能公式

执行一个程序所需的 CPU 时间可以按下式计算：

$$\text{CPU 时间} = \text{IC} \times \text{CPI} \times \text{时钟周期时间}$$

其中，IC 为所执行的指令条数。CPI(Cycles Per Instruction)为每条指令的平均时钟周期数，它可以根据下式计算得出：

$$\text{CPI} = \text{执行程序所需的时钟周期数} / \text{所执行的指令条数}$$

CPU 的性能取决于以下 3 个参数。

(1) 时钟周期时间：取决于硬件实现技术和计算机组成；

(2) CPI：取决于计算机组成和指令系统的结构；

(3) IC：取决于指令系统的结构和编译技术。

CPU 设计中还经常用到下面计算 CPU 时钟周期总数的方法：

$$\text{CPU 时钟} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{IC}_i)$$

其中，n 为指令的种数，IC_i 为第 i 种指令出现的次数，CPI_i 为第 i 种指令所需的平均时钟周期数。这时的 CPU 性能公式为：

$$\text{CPI} = \frac{\text{时钟周期数}}{\text{IC}} = \frac{\sum_{i=1}^{n} (\text{CPI}_i \times \text{IC}_i)}{\text{IC}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{IC}_i}{\text{CI}} \right)$$

$$\text{CPU 时间} = \text{CPU 时钟周期数} \times \text{时钟周期时间} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{IC}_i) \times \text{时钟周期时间}$$

其中，(IC_i/IC)反映了第 i 种指令在所执行的指令总数中所占的比例。

4) 程序的局部性原理

程序的局部性原理是指：程序执行时所访问的存储器地址不是随机分布的，而是相对地簇聚。现在常用的一个经验规则是：程序执行时间的 90% 都是在执行程序中 10% 的代码。

局部性包括时间局部性和空间局部性。时间局部性是指：程序即将用到的信息很可能就是目前正在使用的信息。空间局部性是指：程序即将用到的信息很可能与目前正在使用的信息在空间上相邻或者临近。

2. 计算机系统设计者的主要任务

计算机系统设计者的任务包括指令系统的设计、数据表示的设计、功能的组织、逻辑设计以及其物理实现等。设计一个计算机系统大致要完成以下 3 个方面的工作。

1) 确定用户对计算机系统的功能、价格和性能的要求

总的来说，计算机系统设计者的目标是设计出能满足用户的功能需求、有较长的生命周期且又具有很高的性能价格比的系统。要考虑以下具体的功能需求。

(1) 应用领域。首先要考虑的是：是专用还是通用？是面向科学计算还是面向商用处理？如果是专用，就需要对应用领域进行深入的研究，以确定什么样的计算任务是关键的，

具有什么计算特点等，然后对其进行优化设计。

(2) 软件兼容。软件兼容是指一台计算机上的程序不加修改就可以搬到另一台计算机上正常运行。兼容性方面的考虑对于设计新的计算机系统有很大的影响。

(3) 操作系统需求。

(4) 标准。

2) 软硬件功能分配

软件和硬件在实现功能上是等价的。进行优化设计必须考虑软硬件功能的合理分配。对于任何一种功能来说，用软件实现的优点是设计相对容易、修改简单，而且可以减少硬件成本。但其缺点是所实现的功能的速度较慢。用硬件实现的优点是速度快、性能高，但它修改困难，灵活性差。所以优化设计时要在软硬件之间进行折中和取舍。

3) 设计出生命周期长的系统结构

一个成功的系统结构应该有较长的生命周期，它应该能经得住软、硬件技术的发展和应用的变化。因此，设计者要特别注意计算机应用和计算机技术的发展趋势，设计出具有一定前瞻性的系统结构，以使得它具有较长的生命周期。

3. 计算机系统设计的主要方法

从多级层次结构出发，计算机系统可以有由上往下、由下往上和从中间开始3种不同的设计方法。

1) "由上往下"设计

由上往下设计是先从层次结构中的最上面一级开始，首先确定应用级机器的属性，然后再逐级往下设计，每级都考虑怎样优化上一级的实现。这种方法很适合于专用机的设计，而不适合通用机的设计。

2) "由下往上"设计

这种方法是从层次结构的最下面一级开始，逐层往上设计各层的机器。在硬件技术飞速发展、而软件技术发展相对缓慢的今天，这种设计方法已经难以适应计算机系统的设计要求，很少被采用了。

3) "由中间开始"设计

软、硬件设计分离和脱节是上述"由上往下"和"由下往上"设计方法的主要缺点。要解决这个问题，就必须综合考虑软、硬件的分工，从中间开始设计。这里"中间"是指层次结构中的软硬件的交界面，即在传统机器级与操作系统机器级之间。采用这种方法时，首先要进行软、硬件功能分配，确定好这个界面。然后从这个界面开始，软件设计者开始往上设计操作系统、汇编、编译系统等，硬件设计者开始往下设计传统机器级、微程序机器级、数字逻辑级等。软件和硬件并行设计可以缩短设计周期，设计过程中可以交流协调，是一种交互式的、很好的设计方法。

1.2.3　计算机系统的性能评测

1. 执行时间和吞吐率

如何评测一台计算机的性能，与测试者看问题的角度有关。对于桌面台式计算机来说，

用户关心的是执行单个程序的时间,而对于数据处理中心的管理员来说,他所关心的则是吞吐率。

在比较不同的设计方案时,经常需要对两台计算机的性能进行比较。假设这两台计算机为 X 和 Y,通常"X 的性能是 Y 的 n 倍"是指:

$$n=\frac{\text{执行时间 }Y}{\text{执行时间 }X}=\frac{\frac{1}{\text{性能}_Y}}{\frac{1}{\text{性能}_X}}=\frac{\text{性能}_X}{\text{性能}_Y}$$

目前广泛采用的一致和可靠的性能评价方法,是使用真实程序的执行时间来衡量。

我们常用"CPU 时间"来计算程序的执行时间,"CPU 时间"是指 CPU 执行所给定的程序所花费的时间,不包含 I/O 等待时间以及运行其他程序的时间。CPU 时间还可细分为用户 CPU 时间及系统 CPU 时间,前者表示用户程序所耗费的 CPU 时间,后者表示用户程序执行期间操作系统所耗费的 CPU 时间。

2. 基准测试程序

用于测试和比较性能的基准测试程序的最佳选择是真实应用程序。

为了能比较全面地反映计算机在各个方面的处理性能,通常采用整套测试程序。这组程序称为基准测试程序套件,它是由各种不同的真实应用程序构成的。基准测试程序套件的目标是尽可能全面地反映两台计算机的相对性能。

目前最成功和最常见的测试程序套件是 SPEC 系列,它是由美国的标准性能测试公司(Standard Performance Evaluation Corporation)于 20 世纪 80 年代末创建的。桌面计算机的基准测试程序套件可以分为两大类:处理器性能测试程序和图形性能测试程序。SPEC 最早创建的 SPEC89 是用于测试处理器性能的。SPEC89 后来演化出了 4 个版本:SPEC92、SPEC95、SPEC2000 和 SPEC CPU2006。

3. 性能比较

为了能更好地比较不同计算机的性能,可以采用以下 3 种方法。

1) 总执行时间

我们可以直接用计算机执行所有测试程序的总时间来进行比较,也可以采用平均执行时间来代替总执行时间。平均执行时间是各测试程序执行时间的算术平均值,即:

$$S_m=\frac{1}{n}\sum_{i=1}^{n}T_i$$

其中,T_i 是第 i 个测试程序的执行时间,n 是测试程序组中程序的个数。

如果各程序在测试程序组中所占的比重不同,就可以用加权执行时间来比较。加权执行时间是各测试程序执行时间的加权平均值,即:

$$A_m=\sum_{i=1}^{n}W_i\cdot T_i \tag{1.1}$$

其中,W_i 是第 i 个测试程序在测试程序组中所占的比重,$\sum_{i=1}^{n}W_i=1$。T_i 是该程序的执行时间。

2）调和平均值法

如果性能是用速度（如 MFLOPS）表示，则可以采用调和平均值法进行比较。其公式为：

$$H_m = \frac{n}{\sum_{i=1}^{n} \frac{1}{R_i}} = \frac{n}{\sum_{i=1}^{n} T_i} \tag{1.2}$$

其中，R_i表示执行第 i 个程序的速度，$R_i = 1/T_i$，T_i为第 i 个程序的执行时间。

如果考虑工作负荷中各程序不会以相等比例出现，则可以使用加权调和平均值公式：

$$H_m = \left(\sum_{i=1}^{n} \frac{W_i}{R_i}\right)^{-1} = \left(\sum_{i=1}^{n} W_i T_i\right)^{-1} \tag{1.3}$$

3）几何平均值法

几何平均值法的基本思想来源于性能规格化的方法，即以某台计算机的性能作为参考标准，其他计算机性能则除以该参考标准而获得一个比值。其公式为：

$$G_m = \sqrt[n]{\prod_{i=1}^{n} R_i} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{T_i}} \tag{1.4}$$

式中，R_i表示执行第 i 个程序的速度，$R_i = 1/T_i$，Π为连乘符号。

如果考虑工作负荷中各程序不会以相等比例出现，则可以使用加权几何平均值公式：

$$G_m = \prod_{i=1}^{n} (R_i)^{W_i} = (R_1)^{W_1} \times (R_2)^{W_2} \times \cdots \times (R_n)^{W_n} \tag{1.5}$$

1.2.4　计算机系统结构的发展

1. 冯·诺依曼结构及其改进

最早的存储程序式计算机是美国数学家冯·诺依曼（von Neumann）等人于 1946 年总结并提出来的，它由运算器、控制器、存储器、输入设备和输出设备 5 部分构成。我们经常称之为冯·诺依曼结构计算机。虽然与冯·诺依曼结构相比，现代的计算机系统结构已经发生了很大变化，但就其结构原理来说，占主流地位的仍是改进了的冯·诺依曼结构计算机。

冯·诺依曼结构的主要特点如下。

（1）计算机以运算器为中心，采用集中控制。

（2）在存储器中，指令和数据同等对待。

（3）存储器是按地址访问、按顺序线性编址的一维结构。

（4）指令的执行是顺序的，即一般是按照指令在存储器中存放的顺序执行。

（5）指令由操作码和地址码组成。操作码指明本指令的操作类型，地址码指明操作数地址和存放运算结果的地址。操作数的类型由操作码决定。

（6）指令和数据均以二进制编码表示，采用二进制运算。

后来的计算机针对冯·诺依曼结构的不足之处进行了不断的改进，在系统结构方面有了很大的进展。主要包括以下几个方面。

1）对输入/输出方式的改进

人们先后提出了多种输入/输出方式，包括程序中断、DMA、通道、外围处理机等。这是

把越来越多的输入/输出管理工作从CPU中分离出来,"下放"给新设置的硬件去完成。

2) 采用并行处理技术

在不同的级别采用并行技术,例如微操作级、指令级、线程级、进程级、任务级等。先后出现了向量计算机、阵列处理机、多处理机、MPP(大规模并行处理机)等各种并行处理计算机。

3) 存储器组织结构的发展

在CPU中设置了通用寄存器组,并在CPU和主存之间设置了高速缓冲存储器Cache。

4) 指令系统的发展

发展方向有两个,一个是朝**复杂指令集计算机**(Complex Instruction Set Computer,CISC)的方向发展,把越来越多的功能交由硬件实现。另一个方向是朝精简指令系统的方向发展,只设置使用频度高、功能简单的指令。1979年,D. A. Patterson等人提出了**精减指令集计算机**(Reduced Instruction Set Computer,RISC)的思想。

2. 软件对系统结构的影响

软件对系统结构有多方面的影响。下面只讨论系统结构设计要注意解决的软件可移植性问题。**可移植性**是指一个软件可以不经修改或者只需少量修改就可以由一台计算机移植到另一台计算机上运行。差别只是执行时间的不同。在这种情况下,我们称这两台计算机是**软件兼容**的。实现可移植性的常用方法有3种:统一高级语言,系列机,模拟与仿真。

1) 统一高级语言

如果各计算机能采用同一种高级语言,那么用这种语言编写的应用软件和系统软件的可移植问题就解决了。然而,到目前为止,还没有一种高级语言对各种应用是真正通用的。

2) 系列机

系列机能较好地解决软件开发要求系统结构相对稳定与器件、硬件技术迅速发展的矛盾。系列机的软件兼容有4种:向上兼容,向下兼容,向前兼容,向后兼容。**向上(下)兼容**指的是按某档计算机编制的程序,不加修改就能运行于比它高(低)档的计算机。**向后(前)兼容**是指按某个时期投入市场的某种型号计算机编制的程序,不加修改地就能运行于在它之后(前)投入市场的计算机。向后兼容是肯定要做到的,它是系列机的根本特征。

兼容机:由不同公司厂家生产的具有相同系统结构的计算机。

3) 模拟和仿真

模拟和仿真是实现软件(二进制代码)可移植性的两种常用方法。

模拟是指用软件的方法在一台现有的计算机(称为宿主机Host)上实现另一台计算机(称为虚拟机)的指令系统。通常用解释的方法来实现。除了模拟虚拟机的指令系统外,还要模拟其存储系统、I/O系统、操作系统等。这种方法的缺点是运行速度较慢,性能较差。

仿真是指用一台现有计算机(称为宿主机)上的微程序去解释实现另一台计算机(称为目标机)的指令系统。这个微程序称为仿真微程序。

仿真和模拟的主要区别在于解释执行所用的语言。仿真是用微程序解释执行,而模拟则是用机器语言程序解释执行。因此仿真的运行速度比模拟方法的快,但仿真只能在系统结构差距不大的计算机之间使用。为了取长补短,可以将这两种方法混合使用。

3. 器件发展对系统结构的影响

器件是推动计算机系统结构不断发展的最活跃的因素，这是因为器件是组成计算机系统的最基本的单元。**摩尔定律**指出：集成电路芯片上所集成的晶体管数目每隔 18 个月就翻一番。2003 年以前 CPU 在速度上的提高也是如此，即主频是每隔 18 个月就翻一番。这使得越来越多的功能可以在一块芯片上实现，而且芯片的性能/价格比也越来越高。

高性能、低价格 CPU 芯片的出现使得大规模并行处理系统的实现成为可能。

4. 应用对系统结构的影响

应用对系统结构的发展有着重要的影响。特别是有些领域中的应用问题要求计算机有非常高的计算速度。如果已有的计算机不能满足这些要求，就需要设计和采用新的系统结构。因此，应用需求是促使计算机系统结构发展的最根本的动力。

针对特定的一种应用领域设计的专用计算机的性能往往会比通用计算机高很多。为了满足有些领域的超高速计算性能的要求，往往需要探索和采用新的系统结构，人们往往不惜一切代价来达到其超高性能的目标。过去在这些计算机中所采用的系统结构新技术，后来都逐渐下移到了小型和微型通用计算机中。

1.2.5 计算机系统结构中并行性的发展

1. 并行性的概念

所谓并行性，是指计算机系统在同一时刻或者同一时间间隔内进行多种运算或操作。只要在时间上相互重叠，就存在并行性。它包括同时性与并发性两种含义。

同时性——两个或两个以上的事件在同一时刻发生。

并发性——两个或两个以上的事件在同一时间间隔内发生。

计算机系统中的并行性有不同的等级。从处理数据的角度来看，并行性等级从低到高可分为以下几种。

(1) 字串位串：每次只对一个字的一位进行处理。这是最基本的串行处理方式，不存在并行性。

(2) 字串位并：同时对一个字的全部位进行处理，不同字之间是串行的。具备初步的并行性。这种并行性也称为位级并行。

(3) 字并位串：同时对许多字的同一位(称为位片)进行处理。

(4) 全并行：同时对许多字的全部位或部分位进行处理。这是最高一级的并行。

从执行程序的角度来看，并行性等级从低到高可分为以下 5 种。

1) 指令内部并行

这是指单条指令中各微操作之间的并行。

2) 指令级并行

指令级并行(Instruction Level Parallelism，ILP)是指并行或并发地执行两条或两条以上的指令。

流水线技术使得多条指令能重叠地执行，提高了CPU执行程序的吞吐率。加上RISC方法，计算机达到了每个时钟周期完成一条指令的性能。超标量的方法则更进一步，使得计算机能每个时钟周期启动多条指令，并能由多条流水线在单周期内产生多个运算结果。

3）线程级并行

线程级并行(Thread Level Parallelism，TLP)是指并行执行两个或两个以上的线程。

线程是进程内的一个相对独立的、可独立调度和指派的执行单元。多线程既能提高程序的并发和并行程度，又能减少操作系统的开销。它与多核技术相结合，能达到很好的开发并行性的效果。

4）任务级或过程级并行

这是指并行执行两个或两个以上的过程或任务(程序段)，以子程序或进程为调度单元。

5）作业或程序级并行

这是指并行执行两个或两个以上的作业或程序。

在一个计算机系统中，可以采取多种提高并行性的措施。既可以有数据处理方面的并行性，又可以有执行程序方面的并行性。当并行性提高到一定级别时，则称之为进入并行处理领域。

2. 提高并行性的技术途径

计算机系统中提高并行性的措施有很多，就其基本思想而言，可归纳成以下3条途径。

(1) 时间重叠。在并行性概念中引入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。这种途径原则上不要求重复设置硬件设备。流水线技术是时间重叠的典型实例。

(2) 资源重复。在并行性概念中引入空间因素，以数量取胜。通过重复设置硬件资源，大幅度地提高计算机系统的性能。随着硬件价格的降低，这种途径得到了越来越广泛的应用。

(3) 资源共享。这是一种软件方法，它使多个任务按一定时间顺序轮流使用同一套硬件设备。多道程序、分时系统就是遵循这一途径而产生的。资源共享既降低了成本，又提高了计算机设备的利用率。

在现在的计算机系统中，经常是同时运用时间重叠和资源重复。

3. 单机系统中并行性的发展

在发展高性能单处理机过程中，起主导作用的是时间重叠原理。实现时间重叠的基础是“部件功能专用化”，即把一件工作按功能分割为若干相互联系的部分，把每一部分指定给专门的部件完成，然后按时间重叠原理把各部分的执行过程在时间上重叠起来，使所有部件依次分工完成一组同样的工作。

在单处理机中，资源重复原理的运用也已经十分普遍。例如，多体存储器和多操作部件都是成功应用的结构形式。在多操作部件处理机中，只要指令所需的操作部件空闲，就可以开始执行这条指令(如果操作数已准备好)。这就实现了指令级并行。如果更进一步，设置许多相同的处理单元，让它们在同一个控制器的指挥下，按照同一条指令的要求，对向量或数组的各元素同时进行同一操作，就形成了阵列处理机(有的书中称之为并行处理机)。

4. 多机系统中并行性的发展

多机系统也遵循时间重叠、资源重复、资源共享原理，向着 3 种不同的多处理机方向发展。它们是：同构型多处理机，异构型多处理机和分布式系统。

可以用耦合度来反映多机系统中各计算机之间物理连接的紧密程度和交互作用能力的强弱。**紧密耦合系统**又称直接耦合系统。在这种系统中，计算机之间的物理连接的频带较高，一般是通过总线或高速开关互连，可以共享主存。**松散耦合系统**又称间接耦合系统，一般是通过通道或通信线路实现计算机之间的互连，可以共享外存设备（磁盘、磁带等）。计算机之间的相互作用是在文件或数据集一级上进行。

异构型多处理机系统由多个不同类型、至少担负不同功能的处理机组成，它们按照作业要求的顺序，利用时间重叠原理，依次对它们的多个任务进行加工，各自完成规定的功能动作。**同构型多处理机系统**由多个同类型或至少担负同等功能的处理机组成，它们同时处理同一作业中能并行执行的多个任务。

最早的多机系统并不是为了提高速度，而是为了构成容错系统。如果提高对互联网络的要求，使其具有一定的灵活性、可靠性和可重构性，则可将其发展成一种可重构系统。在这种系统中，平时几台计算机都正常工作，像通常的多处理机系统一样。但一旦发生故障，系统就会重新组织，降低档次继续运行，直到排除故障为止。

5. 并行机的发展变化

(1) 并行机的萌芽阶段(1964—1975 年)；
(2) 向量机的发展和鼎盛阶段(1976—1990 年)；
(3) MPP 出现和蓬勃发展阶段(1990—1995 年)；
(4) 各种体系结构并存阶段(1995—2000 年)；
(5) 机群蓬勃发展阶段(2000 年以后)。

习　　题

1. 概念题

【题 1.1】　解释下列名词

多级层次结构	虚拟机	解释	翻译
计算机系统结构	计算机组成	计算机实现	透明性
系列机	最大并行度	Amdahl 定律	CPI
程序的局部性原理	时间局部性	空间局部性	CISC
RISC	软件兼容	向上兼容	向下兼容
向前兼容	向后兼容	兼容机	摩尔定律
模拟	仿真	并行性	并发性
同时性	时间重叠	资源重复	资源共享

耦合度 紧密耦合系统 松散耦合系统 异构型多处理机系统 同构型多处理机系统

2. 选择题

【题1.2】 直接执行微指令的是( )。
A. 汇编程序 B. 编译程序 C. 硬件 D. 微指令程序
【题1.3】 对汇编语言程序员不透明的是( )。
A. 程序计数器 B. 主存地址寄存器 C. 条件码寄存器 D. 指令寄存器
【题1.4】 "由中间开始设计"的"中间"目前多数是在( )之间。
A. 传统机器级与操作系统之间 B. 传统机器级与微程序级之间
C. 操作系统与汇编语言级之间 D. 微程序级与汇编语言级之间
【题1.5】 最早的冯·诺依曼结构的计算机是以( )为中心的。
A. 运算器 B. 控制器 C. 存储器 D. I/O设备
【题1.6】 从计算机系统结构来看,机器语言程序员看到的机器属性是( )。
A. 计算机软件所要完成的功能 B. 计算机硬件的全部组成
C. 编程要用到的硬件组织 D. 计算机各部件的硬件实现
【题1.7】 不同系列的机器之间,实现可移植性的途径不包括( )。
A. 采用统一的高级语言 B. 采用统一的汇编语言
C. 模拟 D. 仿真
【题1.8】 利用时间重叠原理实现并行处理的是( )。
A. 流水处理机 B. 多处理机 C. 阵列处理机 D. 机群系统
【题1.9】 多处理机实现的并行主要是( )。
A. 指令级并行 B. 任务级并行 C. 操作级并行 D. 操作步骤的并行
【题1.10】 计算机系统结构不包括( )。
A. 信息保护 B. 主存速度 C. 数据表示 D. 机器工作状态

3. 填空题

【题1.11】 常见的计算机系统结构分类法有3种:________、________、________。

【题1.12】 冯氏分类法用系统的最大并行度对计算机进行分类,大多数传统的位并行单处理机属字________位________方式。

【题1.13】 由软件实现的机器称为________。在一个计算机系统中,低层机器的属性对高层机器的程序员往往是________的。

【题1.14】 ________是促使计算机系统结构发展最重要的因素,________是促使计算机系统结构发展最根本的动力,而________是促使计算机系统结构发展最活跃的因素。

【题1.15】 程序的局部性包含程序的________局部性和程序的________局部性。

【题1.16】 从多级层次结构出发,计算机系统可以有________、________和________3种不同的设计方法。

【题1.17】 实现程序可移植性的主要途径有________、________、________和________。

【题 1.18】 为了在不同系统结构的机器之间实现软件移植，可采用________或________方法。

【题 1.19】 软件兼容有________、________、________和________4 种。其中，________是软件兼容的根本特征。

【题 1.20】 广义来说，并行性既包含________性，又包含________性。

【题 1.21】 从执行程序的角度看，并行性等级从低到高可分为________并行、________并行、________并行、________并行和________并行。

【题 1.22】 从处理数据的角度，并行性等级从低到高可以分为________、________、________和________4 种。

【题 1.23】 计算机系统中提高并行性的技术途径有________、________和________三种。在高性能单处理机的发展中，起主导作用的是________这个途径，它的实现基础是________。

【题 1.24】 多机系统的耦合度可以分为________和________。

【题 1.25】 同构型多处理机和异构型多处理机所采用的提高并行性的技术途径分别是________和________。

4. 问答题

【题 1.26】 试用实例说明计算机系统结构、计算机组成与计算机实现之间的相互关系。

【题 1.27】 Flynn 分类法是根据什么对计算机系统的结构进行分类？并分为哪 4 类？

【题 1.28】 冯氏分类法是根据什么对计算机进行分类？把计算机分成哪 4 类？

【题 1.29】 Handler 分类法把计算机的硬件结构分成哪 3 个层次？

【题 1.30】 计算机系统设计中经常使用的 4 个定量原理是什么？并说出它们的含义。

【题 1.31】 计算机系统"由中间开始设计"，其"中间"指的是什么地方？这样设计的好处是什么？

【题 1.32】 硬件和软件在什么意义上是等效的？在什么意义上是不等效的？

【题 1.33】 存储程序计算机在系统结构上的主要特点是什么？

【题 1.34】 系列机概念对计算机发展有什么意义？系列机软件兼容的基本要求是什么？列出几个你所熟知的系列机。

5. 应用题

【题 1.35】 假设有一个计算机系统分为两级，第一级指令比第二级指令在功能上强 X 倍，即一条第一级指令能够完成 X 条第二级指令的工作，且一条第一级指令需要 Y 条第二级指令解释。对于一段在第一级执行时间为 N 的程序，在第二级上的一段等效程序需要执行多少时间？

【题 1.36】 某台主频为 400MHz 的计算机执行标准测试程序，程序中指令类型、执行数量和平均时钟周期数如表 1.1 所示。

表1.1 指令类型、执行数量和平均时钟周期数

指令类型	指令执行数量	平均时钟周期数
整数	45 000	1
数据传送	75 000	2
浮点	8000	4
分支	1500	2

求该计算机的有效CPI、MIPS和程序执行时间。

【题1.37】 假设处理机的时钟频率为40MHz,运行的测试程序共有200 000条指令,由4类指令组成。已知各类指令的CPI和各类指令条数的比例如表1.2所示。

表1.2 各类指令的CPI和各类指令条数的比例

指令类型	CPI	指令条数比例
算术逻辑运算指令	1	60%
Cache命中的load/store指令	2	18%
转移类指令	4	12%
Cache不命中的load/store指令	8	10%

计算处理机运行该测试程序的CPI和MIPS。

【题1.38】 已知4个程序在3台计算机上的执行时间如表1.3所示。

表1.3 执行时间

程序	执行时间		
	计算机A	计算机B	计算机C
程序1	1	10	20
程序2	1000	100	20
程序3	500	1000	50
程序4	100	800	100

假设4个程序都执行100 000 000条指令,计算这3台计算机中每台机器上每个程序的MIPS速率。分别计算它们的算术平均值、几何平均值和调和平均值。

【题1.39】 计算机系统有3个部件可以改进,这3个部件的加速比如下:

部件加速比 $S_1=30$;部件加速比 $S_2=20$;部件加速比 $S_3=10$

(1) 如果部件1和部件2的可改进比例为30%,那么当部件3的可改进比例为多少时,系统的加速比才可以达到10?

(2) 如果3个部件的可改进比例分别为30%、30%和20%,3个部件同时改进,那么系统中不可改进部分的执行时间在总执行时间中占的比例是多少?

【题1.40】 将计算机系统中某一功能的处理速度加快20倍,但该功能的处理时间仅占整个系统运行时间的40%,则采用此增强功能方法后,能使整个系统的性能提高多少?

【题1.41】 某计算机系统采用浮点运算部件后,使浮点运算速度提高到原来的20倍,而系统运行某一程序的整体性能提高到原来的5倍,试计算该程序中浮点操作所占的比例。

【题1.42】 假设我们考虑条件分支指令的两种不同设计方法:

(1) CPU_A:通过比较指令设置条件码,然后测试条件码进行分支。

(2) CPU_B：在分支指令中包括比较过程。

在这两种 CPU 中，条件分支指令都占用 2 个时钟周期，而所有其他指令占用 1 个时钟周期。对于 CPU_A，执行的指令中分支指令占 20%；由于每条分支指令之前都需要有比较指令，因此比较指令也占 20%。由于 CPU_A 在分支时不需要比较，因此 CPU_B 的时钟周期时间是 CPU_A 的 1.25 倍。问：哪一个 CPU 更快？如果 CPU_B 的时钟周期时间只是 CPU_A 的 1.1 倍，哪一个 CPU 更快呢？

【题 1.43】 假设某应用程序中有 4 类操作，通过改进，各操作获得不同的性能提高。具体数据如表 1.4 所示。

表 1.4 各操作获得的性能提高

操作类型	程序中的数量（百万条指令）	改进前的执行时间（周期）	改进后的执行时间（周期）
操作 1	10	2	1
操作 2	30	20	15
操作 3	35	10	3
操作 4	15	4	1

(1) 改进后，各类操作的加速比分别是多少？

(2) 各类操作单独改进后，程序获得的加速比分别是多少？

(3) 4 类操作均改进后，整个程序的加速比是多少？

【题 1.44】 某台处理机的时钟频率为 15MHz，执行测试程序的速率为 10MIPS，假设每次存储器存取需 1 个时钟周期的时间。

(1) 求处理机的 CPI 值。

(2) 假设将处理机的时钟频率提高到 30MHz，但存储器的工作速率不变，这使得每次存储器存取需 2 个时钟周期。如果测试程序中 30%的指令需要 1 次访存，5%的指令需要 2 次访存，其他指令不需要访存，求该测试程序在改进后的处理机上执行的 MIPS。

【题 1.45】 假设浮点数指令 FP 的比例为 30%，其中，浮点数平方根 FPSQR 占全部指令的比例为 4%，FP 操作的 CPI 为 5，FPSQR 操作的 CPI 为 20，其他指令的平均 CPI 为 1.25。现有两种改进方案，第一种是把 FPSQR 操作的 CPI 减至 3，第二种是把所有的 FP 操作的 CPI 减至 3，试比较两种方案对系统性能的提高程度。

【题 1.46】 若在整个测试程序的执行时间中，求浮点数平方根 FPSQR 的操作占 10%。现有两种改进方案：

(1) 采用 FPSQR 硬件，使其速度加快到 10 倍；

(2) 使所有浮点数指令 FP 速度加快到 4 倍，并设 FP 指令占整个程序执行时间的 40%。试比较两种方案的优劣。

题 解

1. 概念题

【题 1.1】 解释下列名词

多级层次结构 —— 按照计算机语言从低级到高级的次序，把计算机系统按功能划分成

多级层次结构，每一层以一种不同的语言为特征。这些层次依次为：微程序机器级，传统机器语言机器级，汇编语言机器级，高级语言机器级，应用语言机器级等。

虚拟机——用软件实现的机器。

解释——对于高一级机器上的程序中的每一条语句或指令，都是转去执行低一级机器上的一段等效程序。执行完后，再去高一级机器取下一条语句或指令，再进行解释执行，如此反复，直到解释执行完整个程序。

翻译——先用转换程序把高一级机器上的程序转换为低一级机器上等效的程序，然后再在这低一级机器上运行，实现程序的功能。

计算机系统结构——指传统机器语言程序员所看到的计算机属性，即概念性结构与功能特性。

计算机组成——指的是计算机系统结构的逻辑实现，包含物理机器级中的数据流和控制流的组成以及逻辑设计等。

计算机实现——指的是计算机组成的物理实现，包括处理机、主存等部件的物理结构，器件的集成度和速度，模块、插件、底板的划分与连接，信号传输，电源、冷却及整机装配技术等。

透明性——在计算机技术中，把这种本来存在的事物或属性，但从某种角度看又好像不存在的概念称为透明性。

系列机——由同一厂家生产的具有相同系统结构，但具有不同组成和实现的一系列不同型号的计算机。

最大并行度——计算机系统在单位时间内能够处理的最大的二进制位数。

Amdahl定律——当对一个系统中的某个部件进行改进后，所能获得的整个系统性能的提高，受限于该部件的执行时间占总执行时间的百分比。

CPI——每条指令的平均执行时钟周期数。

程序的局部性原理——程序执行时所访问的存储器地址不是随机分布的，而是相对地簇聚。

时间局部性——程序即将用到的信息很可能就是目前正在使用的信息。

空间局部性——程序即将用到的信息很可能与目前正在使用的信息在空间上相邻或者临近。

CISC——复杂指令集计算机。

RISC——精减指令集计算机。

软件兼容——一个软件可以不经修改或者只需少量修改就可以由一台计算机移植到另一台计算机上运行。差别只是执行时间的不同。

向上兼容——按某档计算机编制的程序，不加修改就能运行于比它高档的计算机。

向下兼容——按某档计算机编制的程序，不加修改就能运行于比它低档的计算机。

向前兼容——按某个时期投入市场的某种型号计算机编制的程序，不加修改地就能运行于在它之前投入市场的计算机。

向后兼容——按某个时期投入市场的某种型号计算机编制的程序，不加修改地就能运行于在它之后投入市场的计算机。

兼容机——由不同公司厂家生产的具有相同系统结构的计算机。

摩尔定律——摩尔定律指出集成电路芯片上所集成的晶体管数目每隔 18 个月就翻一番。

模拟——用软件的方法在一台现有的计算机（称为宿主机 Host）上实现另一台计算机（称为虚拟机）的指令系统。

仿真——用一台现有计算机（称为宿主机）上的微程序去解释实现另一台计算机（称为目标机）的指令系统。

并行性——计算机系统在同一时刻或者同一时间间隔内进行多种运算或操作。只要在时间上相互重叠，就存在并行性。它包括同时性与并发性两种含义。

并发性——两个或两个以上的事件在同一时间间隔内发生。

同时性——两个或两个以上的事件在同一时刻发生。

时间重叠——在并行性概念中引入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。

资源重复——在并行性概念中引入空间因素，以数量取胜。通过重复设置硬件资源，大幅度地提高计算机系统的性能。

资源共享——这是一种软件方法，它使多个任务按一定时间顺序轮流使用同一套硬件设备。

耦合度——反映多机系统中各计算机之间物理连接的紧密程度和交互作用能力的强弱。

紧密耦合系统——又称直接耦合系统。在这种系统中，计算机之间的物理连接的频带较高，一般是通过总线或高速开关互连，可以共享主存。

松散耦合系统——又称间接耦合系统，一般是通过通道或通信线路实现计算机之间的互连，可以共享外存设备（磁盘、磁带等）。计算机之间的相互作用是在文件或数据集一级上进行。

异构型多处理机系统——由多个不同类型、至少担负不同功能的处理机组成，它们按照作业要求的顺序，利用时间重叠原理，依次对它们的多个任务进行加工，各自完成规定的功能动作。

同构型多处理机系统——由多个同类型或至少担负同等功能的处理机组成，它们同时处理同一作业中能并行执行的多个任务。

2. 选择题

【题 1.2】　答：C
【题 1.3】　答：C
【题 1.4】　答：A
【题 1.5】　答：A
【题 1.6】　答：C
【题 1.7】　答：B
【题 1.8】　答：A
【题 1.9】　答：B
【题 1.10】　答：B

3. 填空题

【题1.11】 答：Flynn分类法、冯氏分类法、Handler分类法

【题1.12】 答：串、并

【题1.13】 答：虚拟机器、透明

【题1.14】 答：软件、应用、器件

【题1.15】 答：时间、空间

【题1.16】 答：由上往下设计、由下往上设计、从中间开始设计

【题1.17】 答：统一高级语言、系列机、模拟、仿真

【题1.18】 答：模拟、仿真

【题1.19】 答：向上兼容、向下兼容、向前兼容、向后兼容、向后兼容

【题1.20】 答：同时、并发

【题1.21】 答：指令内部、指令级、线程级并行、任务级或过程级、作业或程序级

【题1.22】 答：字串位串、字串位并、字并位串、全并行

【题1.23】 答：时间重叠、资源重复、资源共享、时间重叠、部件功能专用化

【题1.24】 答：松散耦合、紧密耦合

【题1.25】 答：资源重复、时间重叠

4. 问答题

【题1.26】 答：如在设计主存系统时，确定主存容量、编址方式、寻址范围等属于计算机系统结构；确定主存周期、逻辑上是否采用并行主存、逻辑设计等属于计算机组成，而选择存储芯片类型、微组装技术、线路设计等属于计算机实现。

计算机组成是计算机系统结构的逻辑实现。计算机实现是计算机组成的物理实现。一种系统结构可以有多种组成。一种组成可以有多种实现。

【题1.27】 答：Flynn分类法是按照指令流和数据流的多倍性进行分类。把计算机系统的结构分为：

(1) 单指令流单数据流SISD

(2) 单指令流多数据流SIMD

(3) 多指令流单数据流MISD

(4) 多指令流多数据流MIMD

【题1.28】 答：冯氏分类法是用系统的最大并行度对计算机进行分类。把计算机分成4类具有不同最大并行度的计算机系统结构。

(1) 字串位串：这是第一代计算机发展初期的纯串行计算机。

(2) 字串位并：这是传统的单处理机，同时处理单个字的多个位。

(3) 字并位串：同时处理多个字的同一位(位片)。

(4) 字并位并：同时处理多个字的多个位。

【题1.29】 答：Handler分类法把计算机的硬件结构分成3个层次，并考虑它们的可并行-流水处理程度。这3个层次是：

(1) 程序控制部件(PCU)的个数k；

(2) 算术逻辑部件(ALU)或处理部件(PE)的个数 d；

(3) 每个算术逻辑部件包含基本逻辑线路(ELC)的套数 w。

【题 1.30】 答：(1) 以经常性事件为重点。在计算机系统的设计中，对经常发生的情况，赋予它优先的处理权和资源使用权，以得到更多的总体上的改进。

(2) Amdahl 定律。加快某部件执行速度所获得的系统性能加速比，受限于该部件在系统中所占的重要性。

(3) CPU 性能公式。

执行一个程序所需的 CPU 时间＝IC×CPI×时钟周期时间

(4) 程序的局部性原理。程序在执行时所访问地址的分布不是随机的，而是相对地簇聚。

【题 1.31】 答："中间"指的是多级层次结构中的软硬件交界面，即传统机器级与操作系统机器级之间。

这样设计，能合理地进行软硬件的功能分配，优化软硬件设计，可为软件和应用提供更多更好的支持；软件和硬件并行设计可以缩短设计周期。

【题 1.32】 答：硬件和软件在实现逻辑功能上是等效的。在原理上，用软件实现的功能完全可以用硬件或固件来实现；用硬件实现的功能也可以用软件进行模拟完成。

但在速度、价格、实现的难易程度上是不同的。对于任何一种功能来说，用软件实现的优点是设计容易、修改简单，而且可以减少硬件成本。但其缺点是所实现的功能的速度较慢。用硬件实现的优点是速度快、性能高，但它修改困难，灵活性差。

【题 1.33】 答：①机器以运算器为中心。②采用存储程序原理。程序(指令)和数据放在同一存储器中，并且没有对两者加以区分。指令和数据一样可以送到运算器进行运算，即由指令组成的程序自身是可以修改的。③存储器是按地址访问的、线性编址的空间。④控制流由指令流产生。⑤指令由操作码和地址码组成。操作码指明本指令的操作类型，地址码指明操作数和操作结果的地址。⑥数据以二进制编码表示，采用二进制运算。

【题 1.34】 答：系列机可以实现系统结构相同的计算机之间的软件移植。它较好地解决了软件开发要求系统结构相对稳定与器件、硬件技术迅速发展的矛盾。

系列机软件兼容的基本要求是保证向后兼容，力争向上兼容。

IBM 公司的 IBM370 系列，Intel 公司的 x86 系列都是较典型的系列机。

5. 应用题

【题 1.35】

解：假设在第一级上用时间 N 执行了该级 IC 条指令。

对第二级而言，为了完成 IC 条指令的功能，第二级指令的条数为 IC/X。

为了执行第二级 IC/X 条指令，需要执行 $(\mathrm{IC}/X)\times Y$ 条第一级的指令对其进行解释，所以对于第二级而言，等效程序的执行时间是：

$$T_2=\left(\frac{\mathrm{IC}}{X}\times X+\frac{\mathrm{IC}}{X}\times Y\right)\times\frac{N}{\mathrm{IC}}=\left(1+\frac{Y}{X}\right)\times N$$

【题 1.36】

解：(1) CPI＝(45 000×1＋75 000×2＋8000×4＋1500×2)/129 500＝1.776

(2) MIPS 速率 $=f/\mathrm{CPI}=400/1.776=225.225$MIPS

(3) 程序执行时间 $=(45\,000\times1+75\,000\times2+8000\times4+1500\times2)/400=575$s

【题 1.37】

解：$\mathrm{CPI}=\sum(\mathrm{CPI}_i\times \mathrm{IC}_i/\mathrm{IC})=1\times0.6+2\times0.18+4\times0.12+8\times0.1=2.24$

$$\mathrm{MIPS}=\frac{\text{时钟频率}}{\mathrm{CPI}\times10^6}=\frac{40\times10^6}{2.24\times10^6}=17.86$$

【题 1.38】

解：MIPS 速率如表 1.5 所示。

表 1.5　MIPS 速率

程　序	MIPS 速率		
	计算机 A	计算机 B	计算机 C
程序 1	100	10	5
程序 2	0.1	1	5
程序 3	0.2	0.1	2
程序 4	1	0.125	1

它们的算术平均值、几何平均值和调和平均值如表 1.6 所示。

表 1.6　算术平均值、几何平均值和调和平均值

程　序	MIPS 速率		
	计算机 A	计算机 B	计算机 C
算术平均值	25.3	2.81	3.25
几何平均值	1.19	0.59	2.66
调和平均值	0.25	0.2	2.1

【题 1.39】

解：(1) 在多个部件可改进的情况下，Amdahl 定理可扩展为：

$$S_n=\frac{1}{\left(1-\sum F_i\right)+\sum\frac{F_i}{S_i}}$$

已知 $S_1=30$，$S_2=20$，$S_3=10$，$S_n=10$，$F_1=0.3$，$F_2=0.3$，得：

$$10=\frac{1}{1-(0.3+0.3+F_3)+(0.3/30+0.3/20+F_3/10)}$$

得 $F_3=0.36$，即部件 3 的可改进比例为 36%。

(2) 设系统改进前的执行时间为 T，则 3 个部件改进前的执行时间为：$(0.3+0.3+0.2)T=0.8T$，不可改进部分的执行时间为 $0.2T$。

已知 3 个部件改进后的加速比分别为 $S_1=30$，$S_2=20$，$S_3=10$，因此 3 个部件改进后的执行时间为：

$$T_n'=\frac{0.3T}{30}+\frac{0.3T}{20}+\frac{0.2T}{10}=0.045T$$

改进后整个系统的执行时间为：$T_n=0.045T+0.2T=0.245T$

那么系统中不可改进部分的执行时间在总执行时间中占的比例是：

$$\frac{0.2T}{0.245T}=0.82$$

【题 1.40】

解：由题可知：可改进比例＝40%＝0.4　部件加速比＝20

根据 Amdahl 定律可知：

$$系统加速比=\frac{1}{(1-0.4)+\frac{0.4}{20}}=1.613$$

采用此增强功能方法后，能使整个系统的性能提高到原来的 1.613 倍。

【题 1.41】

解：由题可知：部件加速比＝20，系统加速比＝5，

根据 Amdahl 定律可知：

$$5=\frac{1}{(1-可改进比例)+\frac{可改进比例}{20}}$$

由此可得：可改进比例＝84.2%

即程序中浮点操作所占的比例为 84.2%。

【题 1.42】

解：我们不考虑所有系统问题，所以可用 CPU 性能公式。占用 2 个时钟周期的分支指令占总指令的 20%，剩下的指令占用 1 个时钟周期。所以

$$CPI_A=0.2\times2+0.80\times1=1.2$$

则 CPU_A 性能为：

$$总CPU时间_A=IC_A\times1.2\times时钟周期_A$$

根据假设，有：

$$时钟周期_B=1.25\times时钟周期_A$$

在 CPU_B 中没有独立的比较指令，所以 CPU_B 的程序量为 CPU_A 的 80%，分支指令的比例为：

$$20\%/80\%=25\%$$

这些分支指令占用 2 个时钟周期，而剩下的 75% 的指令占用 1 个时钟周期，因此：

$$CPI_B=0.25\times2+0.75\times1=1.25$$

因为 CPU_B 不执行比较，故：

$$IC_B=0.8\times IC_A$$

因此 CPU_B 性能为：

$$\begin{aligned}总CPU时间_B&=IC_B\times CPI_B\times时钟周期_B\\&=0.8\times IC_A\times1.25\times(1.25\times时钟周期_A)\\&=1.25\times IC_A\times时钟周期_A\end{aligned}$$

在这些假设之下，尽管 CPU_B 执行指令条数较少，CPU_A 因为有着更短的时钟周期，所以比 CPU_B 快。

如果 CPU_B 的时钟周期时间仅仅是 CPU_A 的 1.1 倍，则

$$时钟周期_B = 1.10 \times 时钟周期_A$$

CPU_B的性能为：

$$\begin{aligned} 总CPU时间_B &= IC_B \times CPI_B \times 时钟周期_B \\ &= 0.8 \times IC_A \times 1.25 \times (1.10 \times 时钟周期_A) \\ &= 1.10 \times IC_A \times 时钟周期_A \end{aligned}$$

因此CPU_B由于执行更少指令条数，比CPU_A运行更快。

【题1.43】

解：(1)和(2) 根据Amdahl定律 $S_n = \dfrac{1}{(1-Fe)+\dfrac{Fe}{Se}}$ 可得

表1.7 各类操作的加速比

操作类型	各类操作的指令条数在程序中所占的比例 F_i	各类操作的加速比 S_i	各类操作单独改进后，程序获得的加速比
操作1	11.1%	2	1.06
操作2	33.3%	1.33	1.09
操作3	38.9%	3.33	1.37
操作4	16.7%	4	1.14

(3) 4类操作均改进后，整个程序的加速比：

$$S_n = \frac{1}{\left(1-\sum F_i\right)+\sum \dfrac{F_i}{S_i}} \approx 2.16$$

【题1.44】

解：(1) 由MIPS=时钟频率/(CPI×10^6)，得处理机的CPI为：

$$CPI_1 = \frac{时钟频率}{MIPS_1 \times 10^6} = \frac{15 \times 10^6}{10 \times 10^6} = 1.5$$

(2) 处理机改进后，程序中30%的指令需要2个时钟周期，5%的指令需要4个时钟周期，另外65%的指令不需访存，其指令的平均时钟周期数仍为1.5个时钟周期，因此，可得测试程序在改进后的处理机上执行的CPI为：

$$CPI_2 = \sum (CPI_i \times IC_i/IC) = 2 \times 0.3 + 4 \times 0.05 + 1.5 \times 0.65 = 1.775$$

测试程序在改进后的处理机上的执行速率为：

$$MIPS_2 = \frac{f_2}{CPI_2 \times 10^6} = \frac{30 \times 10^6}{1.775 \times 10^6} = 16.9$$

【题1.45】

解：没有改进之前，每条指令的平均时钟周期CPI为：

$$CPI = \sum_{i=1}^{n}\left(CPI_i \times \frac{IC_i}{IC}\right) = (5 \times 30\%) + (1.25 \times 70\%) = 2.38$$

(1) 采用第一种方案：FPSQR操作的CPI由$CPI_{FPSQR}=20$减至$CPI'_{FPSQR}=3$，则整个系统的指令平均时钟周期数为：

$$CPI_1 = CPI - (CPI_{FPSQR} - CPI'_{FPSQR}) \times 4\% = 2.38 - (20 - 3) \times 4\% = 1.7$$

(2) 采用第二种方案：所有 FP 操作的 CPI 由 $CPI_{FP}=5$ 减至 $CPI'_{FP}=3$，则整个系统的指令平均时钟周期数为：

$$CPI_2 = CPI - (CPI_{FP} - CPI'_{FP}) \times 30\% = 2.38 - (5-3) \times 30\% = 1.78$$

从降低整个系统的指令平均时钟周期数的程度来看，第一种方案优于第二种方案。

【题 1.46】

解：(1) Fe=0.1，Se=10，则：

$$S_n = \frac{1}{(1-0.1)+\frac{0.1}{10}} \approx 1.10$$

(2) Fe=0.4，Se=4，则：

$$S_n = \frac{1}{(1-0.4)+\frac{0.4}{4}} \approx 1.43$$

比较结果得：方案(2)更优。

第 2 章　指令系统的设计

2.1　基本要求与难点

2.1.1　基本要求

(1) 掌握有关指令系统设计的基本概念。

(2) 理解堆栈型结构、累加器型结构和通用寄存器型结构三者的特点和不同，掌握 3 种通用寄存器型结构的优缺点。

(3) 掌握指令系统设计的基本原则。

(4) 熟练掌握哈夫曼编码、定长编码和等长扩展码这 3 种操作码编码方法，掌握平均码长的计算方法，掌握指令字格式优化的方法。

(5) 理解指令系统的发展和改进。

(6) 了解 CISC 存在的问题，理解设计 RISC 计算机的一般原则。

(7) 理解 MIPS 指令系统结构，熟练掌握其 3 种指令格式，并熟悉其常用的指令。

2.1.2　难点

(1) 指令系统设计的基本原则。

(2) 指令操作码的优化编码和指令字格式优化的方法。

2.2　知识要点

2.2.1　指令系统结构的分类

CPU 中用来存放操作数的存储单元主要有 3 种：堆栈，累加器，通用寄存器组。据此，可以把指令系统的结构分为堆栈型结构、累加器型结构以及通用寄存器型结构。在通用寄存器型结构中，根据 ALU 指令中操作数的来源不同，又可以进一步分为**寄存器-存储器型结构**(简称 RM 结构)和**寄存器-寄存器型结构**(RR 结构)。RM 结构的操作数可以来自存储器，而 RR 结构的操作数则都是来自通用寄存器组。由于在 RR 结构中，只有 load 指令和 store 指令能够访问存储器，所以也称之为 **load-store 结构**。

对于不同类型的结构，指令系统中操作数的位置、个数，以及操作数的给出方式(显式或

隐式)是不同的。显式给出是用指令字中的操作数字段给出,隐式给出则是使用事先约定好了的单元。在**堆栈型结构**中,操作数都是隐式的,即堆栈的栈顶和次栈顶中的数据,运算后结果写入栈顶。在**累加器型结构**中,其一个操作数是隐式的,即累加器,另一个操作数则是显式给出,是一个存储器单元。运算结果送回累加器。在**通用寄存器型结构**中,所有操作数都是显式地给出,它们或者都是来自通用寄存器组,或者是有一个操作数来自存储器。运算结果写入通用寄存器组。

可以看出,在堆栈型或累加器型的机器中,指令字比较短,程序占用的空间比较小。但是,由于在堆栈型机器中不能随机地访问堆栈,难以生成有效的代码,而且对栈顶的访问是个瓶颈。在累加器型的机器中,由于只有一个中间结果暂存器(即累加器),所以需要频繁地访问存储器。

早期的大多数机器都是采用堆栈型结构或累加器型结构,但是自 1980 年以后,大多数机器都陆续采用了通用寄存器型结构。这主要是因为通用寄存器型结构在灵活性和提高性能方面有明显的优势。主要体现在:

(1) 寄存器的访问速度比存储器快很多。

(2) 对编译器而言,能更加容易、有效地分配和使用寄存器。

(3) 寄存器可以用来存放变量。这能带来许多好处:①由于寄存器比存储器快,所以将变量分配给寄存器能加快程序的执行速度;②能够减少对存储器的访问;③可以用更少的地址位来对寄存器进行寻址,从而有效地减少程序的目标代码所占用的空间。

2.2.2　寻址方式

寻址方式是指指令系统中如何形成所要访问的数据的地址。一般来说,寻址方式可以指明指令中的操作数是一个常数、一个寄存器操作数或者是一个存储器操作数。对于存储器操作数来说,由寻址方式确定的存储器地址称为有效地址。

在 CISC 计算机中,寻址的方式比较多,如:寄存器寻址、立即数寻址、偏移寻址、寄存器间接寻址、索引寻址、直接寻址或绝对寻址、存储器间接寻址、自增寻址、自减寻址、缩放寻址等。

表示寻址方式的方法有两种,一种是隐含在指令的操作码中,另一种是在指令字中设置专门的寻址字段,用以直接指出寻址方式。

为了避免出现一个信息字被截断存储在两个存储字中的情况,可以要求信息宽度不超过主存宽度的信息必须存放在一个存储字内,不能跨边界。为了实现这一点,就必须做到:信息在主存中存放的起始地址必须是该信息宽度(字节数)的整数倍,即满足以下条件。

字节信息的起始地址为:×…××××

半字信息的起始地址为:×…×××0

单字信息的起始地址为:×…××00

双字信息的起始地址为:×…×000

这就是所谓的信息存储的**整数边界概念**。

2.2.3 指令系统的设计和优化

1. 指令系统设计的基本原则

指令系统是传统机器语言程序设计者所看到的计算机的主要属性，是软、硬件的主要交界面。它在很大程度上决定了计算机具有的基本功能。指令系统的设计包括指令的功能设计和指令格式的设计。

对指令系统的基本要求是完整性、规整性、正交性、高效率和兼容性。

完整性是指在一个有限可用的存储空间内，对于任何可解的问题，编制计算程序时，指令系统所提供的指令足够使用。完整性要求指令系统功能齐全、使用方便。通用计算机系统的基本指令至少包括4类：算术和逻辑运算，数据传输，控制，系统。

规整性主要包括对称性和均匀性。**对称性**是指所有与指令系统有关的存储单元的使用、操作码的设置等都是对称的。**均匀性**是指对于各种不同的操作数类型、字长、操作种类和数据存储单元，指令的设置都要同等对待。

正交性是指在指令中各个不同含义的字段，如操作类型、数据类型、寻址方式字段等，在编码时应互不相关、相互独立。

高效率是指指令的执行速度快、使用频度高。在RISC结构中，大多数指令都能在一个节拍内完成，而且只设置使用频度高的指令。

兼容性主要是要实现向后兼容，指令系统可以增加新指令，但不能删除指令或更改指令的功能。

在设计系统时，有两种截然不同的设计策略：CISC和RISC。CISC的策略是增强指令功能，把越来越多的功能交由硬件来实现，指令的数量也是越来越多。RISC的策略则是尽可能地把指令系统简化，不仅指令的条数少，而且指令的功能也比较简单。

2. 控制指令

控制指令是用来改变控制流的。为便于论述，本书约定：当指令是无条件地改变控制流时，称之为跳转指令；而当控制指令是有条件地改变控制流时，则称之为分支指令。

能够改变控制流的指令有4种：条件分支(conditional branch)、跳转(jump)、过程调用(call)和过程返回(return)。统计数据表明，改变控制流的大部分指令是分支指令。因此，如何表示分支条件就显得非常重要。现在常用的3种表示分支条件的方法及其优缺点见表2.1。

表2.1 表示分支条件的主要方法及其优缺点

名　称	检测分支条件的方法	优　点	缺　点
条件码(CC)	检测由ALU操作设置的一些特殊的位(即CC)	可以自由设置分支条件	条件码是增设的状态。而且它限制了指令的执行顺序，因为它们要保证条件码能顺利地传送给分支指令

续表

名　　称	检测分支条件的方法	优　　点	缺　　点
条件寄存器	比较指令把比较结果放入任何一个寄存器，检测时就检测该寄存器	简单	占用一个寄存器
比较与分支	比较操作是分支指令的一部分，通常这种比较是受到一定限制的	用一条指令（而不是两条）就能实现分支	当采用流水方式时，该指令的操作可能太多，在一拍内做不完

在控制指令中，指定转移目标地址最常用的方法是在指令中提供一个偏移量，由该偏移量和程序计数器（PC）的值相加而得出目标地址。这种寻址方式叫作 PC 相对寻址。

对于过程调用和返回而言，除了要改变控制流之外，可能还要保存机器状态。至少也得保存返回地址，一般是放在专用的链接寄存器或堆栈中。

3. 指令操作码的优化

指令一般由两部分组成：操作码和地址码。指令格式的设计就是确定指令字的编码方式，包括操作码字段和地址码字段的编码和表示方式。指令格式不仅对编译形成的代码的长度有影响，而且对处理器的实现也有影响。

1）哈夫曼编码

哈夫曼压缩概念的基本思想是：当各种事件发生的概率不均等时，可以对发生概率最高的事件用最短的位数（时间）来表示（处理），而对于出现概率较低的事件，则可以用较长的位数（时间）来表示（处理），从而使总的平均位数（时间）缩短。

哈夫曼编码可以通过构造哈夫曼树来求得。

操作码优化的程度可以用信息熵 $H=-\sum_{i=1}^{n} p_i \log_2 p_i$ 来衡量。它表示用二进制编码表示 n 个码点时，理论上的最短平均编码长度。

虽然可以利用哈夫曼编码来减少操作码的平均位数，但所获得的编码是变长度的，不规整，不利于硬件处理。

2）等长扩展码

在早期的计算机上，为了便于分级译码，一般都采用等长扩展码，如 4 8 12 位等。4 8 12 的扩展方法有许多种，例如：15/15/15 法和 8/64/512 法。需要对各扩展方案进行比较，以便找出一种平均码长尽可能短、码长种类个数不能过多的、便于优化实现的方案。

3）定长操作码

随着计算机存储器空间的日益加大，为了保证操作码的译码速度、减少译码的复杂度，现在许多计算机都采用了固定长度的操作码，所有指令的操作码都是同一的长度（如 8 位）。特别是 RISC 结构的机器更是如此。这是以程序的存储空间为代价来换取硬件实现上的好处。

4. 指令字格式的优化

为了能利用操作码缩短所带来的好处，可以采用地址个数可变和/或地址码长度可变的方案。寻址方式的表示方法有两种：与操作码一起编码或设置专门的地址描述符。如果处

理机具有多种寻址方式,而且指令有多个操作数,那么就很难跟操作码一起编码,而是应该给每个操作数分配一个地址描述符,由描述符指出采用什么寻址方式。如果处理机采用load-store结构,寻址方式只有很少几种,那么就可以把寻址方式编码到操作码中。

在采用固定长度操作码的情况下,通过设置不同的地址字段,也可以形成不同特点的编码格式。第一种是可变长度编码格式,当指令系统的寻址方式和操作种类很多时,这种编码格式是最好的。这种方法试图用最少的二进制位来表示目标代码。但是,这种编码格式有可能会使各条指令的字长和执行时间相差很大。

第二种是固定长度编码格式,它将操作类型和寻址方式一起编码到操作码中。采用这种编码格式时,经常是所有指令的长度都是固定统一的。当寻址方式和操作类型非常少时,这种编码格式非常好,它可以有效地降低译码的复杂度,提高译码的速度。大部分RISC的指令系统都采用了这种编码格式。

第三种为混合型编码格式,它是把上述两种方法结合起来,即提供若干种固定的指令字长,以期达到既能够减少目标代码长度,又能降低译码复杂度的目标。

2.2.4 指令系统的发展和改进

1. 沿CISC方向发展和改进指令系统

指令数量多、功能多样是CISC指令系统的一大特点。可以从以下3个方面对CISC指令系统进行改进。

(1) 面向目标程序增强指令功能。

对大量的目标程序及其执行情况进行统计分析,找出那些使用频度高、执行时间长的指令或指令串。对于使用频度高的指令,用硬件加快其执行;对于使用频度高的指令串,用一条新的指令来替代。

(2) 面向高级语言的优化实现来改进指令系统。

大多数高级语言与一般的机器语言的语义差距非常大,这就为高级语言程序的编译带来了一些问题。一方面是编译器本身比较复杂,另一方面是生成的目标代码难以达到很好的优化。因此,需要改进指令系统,增加对高级语言和编译器的支持。可以从以下3个方面入手。

① 针对高级语言中使用频度高、执行时间长的语句,增强有关指令的功能。

② 统计结果表明,条件转移(IF)和无条件转移(GOTO)语句所占的比例也比较高,达到了20%以上,所以增强转移指令的功能,增加转移指令的种类是必要的。

③ 增强系统结构的规整性,减少系统结构中的各种例外情况。

经过上述扩充后,对高级语言的优化实现提供了有力的支持,机器语言和高级语言的语义差距缩小了许多。这样的计算机称为面向高级语言的计算机。

(3) 面向操作系统的优化实现改进指令系统。

系统结构必须对操作系统的实现提供专门的指令。尽管这些指令的使用频度比较低,但如果没有它们的支持,操作系统将无法实现。指令系统往往设置了支持以下操作的指令:①处理机工作状态和访问方式的切换;②进程的管理和切换;③存储管理和信息保护;④进程的同步与互斥,信号灯的管理等。

2. 沿 RISC 方向发展和改进指令系统

从 1979 年开始，美国加州大学 Berkeley 分校以 Patterson 为首的研究小组对指令系统结构的合理性进行了深入研究，他们的研究结果表明，CISC 指令集结构存在以下问题。

(1) 各种指令的使用频度相差悬殊。只有 20%的指令使用频度比较高，占运行时间的 80%。而其余 80%的指令只在 20%的运行时间内才会用到。而且使用频度高的指令也是最简单的指令。

(2) 指令系统庞大，指令条数很多，许多指令的功能又很复杂。这使得控制器硬件变得非常复杂。

(3) 许多指令由于操作繁杂，其 CPI 值比较大，执行速度慢。采用这些复杂指令有可能使整个程序的执行时间反而增加。

(4) 由于指令功能复杂，规整性不好，不利于采用流水技术来提高性能。

Patterson 等人提出了 RISC 指令集结构的设计思想。RISC 是近代计算机系统结构发展史中的一个重要里程碑。

设计 RISC 计算机一般应当遵循以下原则。

(1) 指令条数少、指令功能简单。确定指令系统时，只选取使用频度很高的指令，在此基础上补充一些最有用的指令(如支持操作系统和高级语言实现的指令)。

(2) 采用简单而又统一的指令格式，并减少寻址方式。指令字长都为 32 位或 64 位。

(3) 指令的执行在单周期内完成(采用流水线技术后)。

(4) 采用 load-store 结构。即只有 load 和 store 指令才能访问存储器。

(5) 大多数指令都采用硬连逻辑来实现。

(6) 强调优化编译器的作用，为高级语言程序生成优化的代码。

2.2.5　操作数的类型和大小

计算机系统所能处理的数据类型各式各样。在设计计算机系统结构时，需要研究哪些数据类型用硬件实现，哪些用软件实现，并研究它们的实现方法。

数据表示是指计算机硬件能够直接识别、指令系统可以直接调用的数据类型。它一般是所有数据类型中最常用、相对比较简单、用硬件实现比较容易的几种。例如：定点数(整数)、逻辑数(布尔数)、浮点数(实数)、字符、字符串等。

表示操作数类型的方法有以下两种。

(1) 由指令中的操作码指定操作数的类型。这是最常用的方法。绝大多数机器都采用了这种方法。

(2) 给数据加上标识(tag)，由数据本身给出操作数类型。这就是带标志符的数据表示。硬件通过识别这些标志符就能得知操作数的类型，并进行相应的操作。

带标志符的数据表示有很多优点，但由于需要在执行过程中动态检测标志符，动态开销比较大，所以采用这种方案的机器很少见。

本书中，操作数的大小(size)是指操作数的位数或字节数。一般来说，主要的大小有：字节(8 位)、半字(16 位)、字(32 位)和双字(64 位)。

2.2.6　MIPS 指令系统结构

1981年,Stanford 大学的 Hennessy 及其同事们发表了他们的 MIPS 计算机,后来,在此基础上形成了 MIPS 系列微处理器。到目前为止,已经出现了许多版本的 MIPS。下面将介绍 MIPS64 的一个子集,并将其简称为 MIPS。

1. MIPS 的寄存器

MIPS64 有 32 个 64 位通用寄存器: R0,R1,…,R31。它们被简称为 GPRs(General-Purpose Registers),有时也被称为整数寄存器。R0 的值永远是 0。此外,还有 32 个 64 位浮点数寄存器: F0,F1,…,F31。它们被简称为 FPRs(Floating-Point Registers)。

2. MIPS 的数据表示

MIPS 的数据表示有以下两种。

(1) 整数: 字节(8 位),半字(16 位),字(32 位)和双字(64 位)。

(2) 浮点数: 单精度浮点数(32 位),双精度浮点数(64 位)。

MIPS64 的操作是针对 64 位整数以及 32 位或 64 位浮点数进行的。字节、半字或者字在装入 64 位寄存器时,用零扩展或者用符号位扩展来填充该寄存器的剩余部分。装入以后,对它们将按照 64 位整数的方式进行运算。

3. MIPS 的数据寻址方式

MIPS 的数据寻址方式只有立即数寻址和偏移量寻址两种,立即数字段和偏移量字段都是 16 位的。寄存器间接寻址是通过把 0 作为偏移量来实现的,16 位绝对寻址是通过把 R0(其值永远为 0)作为基址寄存器来完成的。

MIPS 的寻址方式是编码到操作码中的。

MIPS 的存储器是按字节寻址的,地址为 64 位。由于 MIPS 是 load store 结构,GPRs 和 FPRs 与存储器之间的数据传送都是通过 load 和 store 指令来完成的。与 GPRs 有关的存储器访问可以是字节、半字、字或双字。与 FPRs 有关的存储器访问可以是单精度浮点数或双精度浮点数。所有存储器访问都必须是边界对齐的。

4. MIPS 的指令格式

为了使处理器更容易进行流水实现和译码,所有的指令都是 32 位的,其格式见图 2.1。这些指令格式很简单,其中操作码占 6 位。MIPS 按不同类型的指令设置不同的格式,共有 3 种格式,它们分别对应于 I 类指令、R 类指令、J 类指令。在这 3 种格式中,同名字段的位置固定不变。

1) I 类指令

这类指令包括所有的 load 和 store 指令,立即数指令,分支指令,寄存器跳转指令,寄存器链接跳转指令。其格式如图 2.1(a)所示,其中的立即数字段为 16 位,用于提供立即数或偏移量。



图 2.1　MIPS 的指令格式

(1) load 指令：访存有效地址为 Regs[rs]+immediate，从存储器取来的数据放入寄存器 rt。

(2) store 指令：访存有效地址为 Regs[rs]+immediate，要存入存储器的数据放在寄存器 rt 中。

(3) 立即数指令：Regs[rt] ← Regs[rs] op immediate。

(4) 分支指令：转移目标地址为 PC+immediate，Regs[rs]为用于比较的值。

(5) 寄存器跳转、寄存器跳转并链接：转移目标地址为 Regs[rs]。

2) R 类指令

包括 ALU 指令，专用寄存器读/写指令，move 指令等。

ALU 指令：Regs[rd]← Regs[rs] funct Regs[rt]。其中，funct 为具体的运算操作编码。

3) J 类指令

包括跳转指令、跳转并链接指令、自陷指令，异常返回指令。在这类指令中，指令字的低 26 位是偏移量，它与 PC 值相加形成跳转的地址。

5. MIPS 的操作

MIPS 指令可以分为 4 大类：load 和 store，ALU 操作，分支与跳转，浮点操作。

除了 R0 外，所有通用寄存器与浮点寄存器都可以进行 load 或 store。单精度浮点数占用浮点寄存器的一半，单精度与双精度之间的转换必须显式地进行。浮点数的格式是 IEEE 754。

习　题

1. 概念题

【题 2.1】　解释下列名词

堆栈型机器	累加器型机器	通用寄存器型机器	寻址方式
信息存储的整数边界	指令系统的完整性	指令系统的规整性	数据表示
指令系统的正交性	PC相对寻址		

2. 选择题

【题2.2】 不需要编址的数据存储空间是(　　)。

A. CPU中的通用寄存器　　B. 主存储器

C. I/O接口中的寄存器　　D. 堆栈

【题2.3】 信息按整数边界存储的主要优点是(　　)。

A. 访存速度快　　B. 节约主存单元　　C. 指令字的规整化　　D. 指令的优化

【题2.4】 操作码优化的主要目的是(　　)。

A. 缩短指令字长　　B. 减少程序总位数

C. 增加指令字表示的信息　　D. A、B和C

【题2.5】 平均码长最短的编码是(　　)。

A. 定长编码

B. 哈夫曼编码

C. 扩展编码

D. 需要根据编码使用的频度计算平均码长后确定

【题2.6】 2-4扩展编码最多可以得到的码点数是(　　)。

A. 6　　B. 7　　C. 10　　D. 13

【题2.7】 面向目标程序优化的思想是(　　)。

A. 通过使用频度分析来改进指令系统

B. 增设强功能复合指令代替原来的软件实现

C. A和B

D. 面向编译系统改进指令系统

【题2.8】 RISC执行程序的速度比CISC要快的原因是(　　)。

A. RISC的指令系统中指令条数较少

B. 程序在RISC上编译生成的目标程序较短

C. RISC的指令平均执行周期数较少

D. RISC只允许load和store指令访存

【题2.9】 RISC采用寄存器窗口重叠技术,从而大大减少了(　　)。

A. 绝大多数指令的执行时间　　B. 程序调用引起的访存次数

C. 目标程序的指令条数　　D. CPU访存的访问周期

3. 填空题

【题2.10】 CPU中用来存储操作数的存储单元主要有________、________和________。

【题2.11】 可将大多数通用寄存器型指令系统结构分为________、________和________3类。

【题 2.12】　对指令系统的基本要求是：________、________、________、________和________。

【题 2.13】　常用的 3 种表示分支条件的技术是________、________和________。

【题 2.14】　改变控制流程的 4 种情况有________、________、________和________。

【题 2.15】　当控制指令为无条件改变控制流时，称之为________。为有条件改变控制流时，称之为________。

【题 2.16】　2-4-6 扩展编码方法的最短码长是________位，最长码长是________位。最多可编码的码点数为________个。

【题 2.17】　指令系统编码格式有________、________和________3 种。

【题 2.18】　对 CISC 指令系统可以从________、________和________3 个方面进行改进。

4. 简答题

【题 2.19】　区别不同指令系统结构的主要因素是什么？根据这个主要因素可将指令系统结构分为哪 3 类？

【题 2.20】　通用寄存器型指令系统结构在灵活性和提高性能方面的优势主要体现在哪几个方面？

【题 2.21】　常见的 3 种通用寄存器型机器的优缺点各有哪些？

【题 2.22】　计算机指令系统结构设计所涉及的内容有哪些？

【题 2.23】　信息熵 H 的含义是什么？在优化编码中，H 有何作用？

【题 2.24】　简述指令系统结构中采用多种寻址方式的优缺点。

【题 2.25】　表示寻址方式的主要方法有哪些？简述这些方法的优缺点。

【题 2.26】　指令系统的规整性主要包括哪两个方面？简述其含义。

【题 2.27】　简述在指令操作码优化中哈夫曼压缩概念的基本思想。

【题 2.28】　通常有哪几种指令格式？简述其适用范围。

【题 2.29】　简述操作数的类型及其相应的表示方法。

【题 2.30】　数据结构和机器的数据表示之间是什么关系？确定和引入数据表示的基本原则是什么？

【题 2.31】　根据 CPU 性能公式简述 RISC 指令集结构计算机和 CISC 指令集结构计算机的性能特点。

【题 2.32】　从当前的计算机技术观点来看，CISC 结构有什么缺点？

【题 2.33】　简述 RISC 结构的设计原则。

【题 2.34】　试就指令格式、寻址方式和每条指令的周期数(CPI)等方面比较 RISC 和 CISC 处理机的指令系统结构。

5. 应用题

【题 2.35】　某台处理机的各条指令使用频度如表 2.2 所示。

表2.2　各条指令使用频度

指令	使用频度	指令	使用频度	指令	使用频度
ADD	43%	JOM	6%	CIL	2%
SUB	13%	STO	5%	CLA	22%
JMP	7%	SHR	1%	STP	1%

请分别设计这9条指令操作码的哈夫曼编码、3/3/3扩展编码和2/7扩展编码，并计算这3种编码的平均码长。

【题2.36】 某机器的指令字长为16位，设有单地址指令和两地址指令。若每个地址字段均为6位，且两地址指令有A条，问单地址指令最多可以有多少条？

【题2.37】 某处理机的指令系统要求有：三地址指令4条，单地址指令255条，零地址指令16条。设指令字长为12位，每个地址码长度为3位。问能否用扩展编码为其操作码编码？如果要求单地址指令为254条，能否对其操作码扩展编码？说明理由。

【题2.38】 一台模型机共有7条指令，各指令的使用频度分别为：35%(I_1)，25%(I_2)，20%(I_3)，10%(I_4)，5%(I_5)，3%(I_6)，2%(I_7)；有8个通用寄存器和两个变址寄存器。

(1) 设计7条指令操作码的哈夫曼编码，并计算操作码的平均码长。

(2) 若要求设计8位长的寄存器-寄存器型指令3条，16位长的寄存器-存储器型变址寻址指令4条，变址范围为−127～+127，请设计指令格式，并给出指令各字段的长度和操作码编码。

【题2.39】 某处理机的指令字长为16位，有二地址指令、单地址指令和零地址指令3类，每个地址字段的长度均为6位。

(1) 如果二地址指令有15条，单地址指令和零地址指令的条数基本相等，那么，单地址指令和零地址指令各有多少条？为3类指令分配操作码。

(2) 如果指令系统要求这3类指令条数的比例为1∶9∶9，那么，这3类指令各有多少条？为3类指令分配操作码。

【题2.40】 设某台计算机有9条指令，各指令的使用频度分别为：

I_1：52%　　I_2：14%　　I_3：12%　　I_4：7%　　I_5：6%

I_6：5 %　　I_7：2%　　I_8：1%　　I_9：1%

试分别用哈夫曼编码和2 4 6等长扩展编码为其操作码编码，并分别计算平均码长。

【题2.41】 假设某模型机有7条指令，这些指令的使用频度如下所示。

I_1：0.40　　I_2：0.30　　I_3：0.15　　I_4：0.05　　I_5：0.04

I_6：0.03　　I_7：0.03

(1) 计算这7条指令的操作码编码的最短平均码长；

(2) 画出哈夫曼树，写出这7条指令的哈夫曼编码，并计算该编码的平均码长和信息冗余量。

题　　解

1. 概念题

【题 2.1】 解释下列名词

堆栈型机器——CPU 中存储操作数的单元是堆栈的机器。

累加器型机器——CPU 中存储操作数的单元是累加器的机器。

通用寄存器型机器——CPU 中存储操作数的单元是通用寄存器的机器。

寻址方式——指令系统中形成所要访问的数据地址的方法。一般来说，寻址方式可以指明指令中的操作数是一个常数、一个寄存器操作数或者是一个存储器操作数。

信息存储的整数边界——信息在主存中存放的起始地址必须是该信息宽度(字节数)的整数倍。

指令系统的完整性——在一个有限可用的存储空间内，对于任何可解的问题，编制计算程序时，指令系统所提供的指令足够使用。

指令系统的规整性——没有或尽可能减少例外的情况和特殊的应用，所有运算都能对称、均匀地在存储器单元或寄存器单元之间进行。

数据表示——硬件结构能够识别、指令系统可以直接调用的数据类型。

指令系统的正交性——指在指令中各个不同含义的字段，如操作类型、数据类型、寻址方式字段等，在编码时应互不相关、相互独立。

PC 相对寻址——在指令中提供一个偏移量，由该偏移量和程序计数器(PC)的值相加而得出有效地址。

2. 选择题

【题 2.2】 答：D

【题 2.3】 答：A

【题 2.4】 答：D

【题 2.5】 答：B

【题 2.6】 答：D

【题 2.7】 答：C

【题 2.8】 答：C

【题 2.9】 答：B

3. 填空题

【题 2.10】 答：堆栈、累加器、寄存器

【题 2.11】 答：寄存器-寄存器型结构、寄存器-存储器型结构、存储器-存储器型结构

【题 2.12】 答：完整性、规整性、正交性、高效率、兼容性

【题 2.13】 答：条件码、条件寄存器、比较与分支

【题2.14】 答：跳转、分支、过程调用、过程返回

【题2.15】 答：跳转、分支

【题2.16】 答：2、6、46

【题2.17】 答：变长编码格式、固定长度编码格式、混合型编码格式

【题2.18】 答：面向目标程序增强指令功能、面向高级语言的优化实现来改进指令系统、面向操作系统的优化实现改进指令系统

4. 简答题

【题2.19】 答：区别不同指令系统结构的主要因素是CPU中用来存储操作数的存储单元。据此可将指令系统结构分为堆栈结构、累加器结构和通用寄存器结构。

【题2.20】 答：主要体现在：

(1) 寄存器的访问速度比存储器快很多。

(2) 对编译器而言，能更加容易、有效地分配和使用寄存器。

(3) 寄存器可以用来存放变量。这能带来许多好处：①由于寄存器比存储器快，所以将变量分配给寄存器能加快程序的执行速度；②能够减少对存储器的访问；③可以用更少的地址位来对寄存器进行寻址，从而有效地减少程序的目标代码所占用的空间。

【题2.21】 答：常见的3种通用寄存器型机器的优缺点如表2.3所示。

表2.3 常见的3种通用寄存器型机器的优缺点

指令系统结构类型	优 点	缺 点
寄存器-寄存器型	指令字长固定，指令结构简洁，是一种简单的代码生成模型，各种指令的执行时钟周期数相近	与指令中含存储器操作数的指令系统结构相比，指令条数多，目标代码不够紧凑，因而程序占用的空间比较大
寄存器-存储器型	可以在ALU指令中直接对存储器操作数进行引用，而不必先用load指令进行加载。容易对指令进行编码，目标代码比较紧凑	由于有一个操作数的内容将被破坏，所以指令中的两个操作数不对称。在一条指令中同时对寄存器操作数和存储器操作数进行编码，有可能限制指令所能够表示的寄存器个数。指令的执行时钟周期数因操作数的来源(寄存器或存储器)不同而差别比较大
存储器-存储器型	目标代码最紧凑，不需要设置寄存器来保存变量	指令字长变化很大，特别是3操作数指令。而且每条指令完成的工作也差别很大。对存储器的频繁访问会使存储器成为瓶颈。这种类型的指令系统现在已不用了

【题2.22】 答：①指令系统功能设计。主要有RISC和CISC两种技术发展方向。②寻址方式的设计。设置寻址方式可以通过对基准程序进行测试统计，查看各种寻址方式的使用频率，根据使用频率设置必要的寻址方式。③操作数表示和操作数类型。主要的操作数类型和操作数表示的选择有浮点数据类型、整型数据类型、字符型、十进制数据类型等。④寻址方式的表示。可以将寻址方式编码于操作码中，也可以将寻址方式作为一个单独的字段来表示。⑤指令系统格式的设计。有变长编码格式、固定长度编码格式和混合型编码

格式 3 种。

【题 2.23】　答：信息熵 H 的含义是：已知 n 个码点的使用频度 p_i，$i=1,2,\cdots,n$，用 2 进制数对 n 个码点编码的最短平均码长为：

$$H=-\sum_{i=1}^{n} p_i \log_2 p_i$$

各种优化编码方法都可得出实际编码的平均码长 L，可以由 H 得出不同编码的信息冗余量 $R=(L-H)/L$。通过比较 R 的值来衡量不同编码的优劣。

【题 2.24】　答：在指令系统结构中采用多种寻址方式可以显著地减少程序的指令条数。但这同时也可能增加实现的复杂度和使用这些寻址方式的指令的执行时钟周期数(CPI)。

【题 2.25】　答：表示寻址方式有两种常用的方法：①将寻址方式编于操作码中，操作码在描述指令的同时也描述相应的寻址方式。这种方式译码快，但操作码和寻址方式的结合不仅增加了指令的条数，导致了指令的多样性，而且增加了 CPU 对指令译码的难度。②为每个操作数设置一个地址描述符，由该地址描述符表示相应操作数的寻址方式。这种方式译码较慢，但操作码和寻址独立，易于指令扩展。

【题 2.26】　答：规整性主要包括对称性和均匀性。对称性是指所有与指令系统有关的存储单元的使用、操作码的设置等都是对称的。均匀性是指对于各种不同的操作数类型、字长、操作种类和数据存储单元，指令的设置都要同等对待。

【题 2.27】　答：当各种事件发生的概率不均等时，可以对发生概率最高的事件用最短的位数(时间)来表示(处理)，而对于出现概率较低的事件，则可以用较长的位数(时间)来表示(处理)，从而使总的平均位数(时间)缩短。

【题 2.28】　答：①变长编码格式。如果系统结构设计者感兴趣的是程序的目标代码大小，而不是性能，就可以采用变长编码格式。②固定长度编码格式。如果感兴趣的是性能，而不是程序的目标代码大小，则可以选择固定长度编码格式。③混合型编码格式。需要兼顾降低目标代码长度和降低译码复杂度时，可以采用混合型编码格式。

【题 2.29】　答：操作数的类型主要有整数(定点)、浮点、十进制、字符、字符串、向量、堆栈等。操作数类型有两种表示方法：①操作数的类型由操作码的编码指定，这是最常见的一种方法；②数据可以附上由硬件解释的标记，由这些标记指定操作数的类型，从而选择适当的运算。

【题 2.30】　答：数据结构是指由软件进行处理和实现的各种数据类型。数据结构研究的是这些数据类型的逻辑结构与物理结构之间的关系，并给出相应的算法。数据表示是指计算机硬件能够直接识别、指令系统可以直接调用的数据类型。它一般是所有数据类型中最常用、相对比较简单、用硬件实现比较容易的几种。

确定和引入数据表示的基本原则：①系统的效率是否提高，是否减少了实现时间和存储空间；②通用性和利用率是否提高。

【题 2.31】　答：CPU 性能公式：CPU 时间＝IC×CPI×T

其中，IC 为目标程序被执行的指令条数，CPI 为指令平均执行周期数，T 是时钟周期的时间。

相同功能的CISC目标程序的指令条数IC_{CISC}少于RISC的IC_{RISC}，但是CISC的CPI_{CISC}和T_{CISC}都大于RISC的CPI_{RISC}和T_{RISC}，因此，CISC目标程序的执行时间比RISC的更长。

【题2.32】 答：①CISC结构的指令系统中，各种指令的使用频率相差悬殊。②CISC结构指令系统的复杂性带来了计算机系统结构的复杂性，这不仅增加了研制时间和成本，而且容易造成设计错误。③CISC结构指令系统的复杂性给VLSI设计增加了很大负担，不利于单片集成。④CISC结构的指令系统中，许多复杂指令需要很复杂的操作，因而运行速度慢。⑤在CISC结构的指令系统中，由于各条指令的功能不均衡性，不利于采用先进的计算机系统结构技术(如流水技术)来提高系统的性能。

【题2.33】 答：①选取使用频率最高的指令，并补充一些最有用的指令；②每条指令的功能应尽可能简单，并在一个机器周期内完成；③所有指令长度均相同；④只有load和store操作指令才访问存储器，其他指令操作均在寄存器之间进行；⑤以简单有效的方式支持高级语言。

【题2.34】 答：比较结果见表2.4。

表2.4　CISC和RISC的比较

比较内容	CISC	RISC
指令格式	变长编码	定长编码
寻址方式	各种都有	只有load/store指令可以访存
CPI	远远大于1	为1

5. 应用题

【题2.35】

解：由给出指令的使用频度，得到两种哈夫曼树如图2.2和图2.3所示。

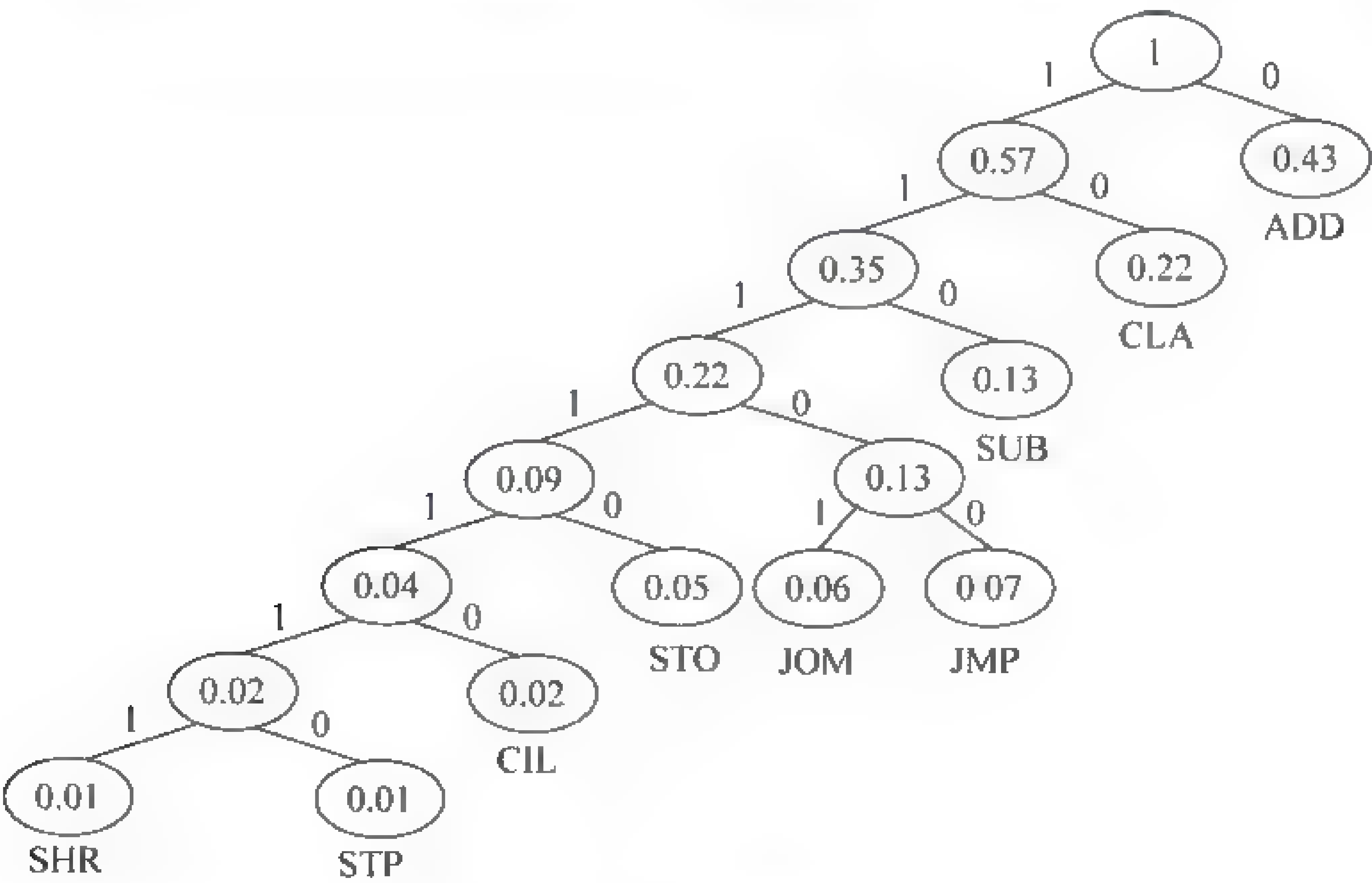


图2.2　哈夫曼树(1)

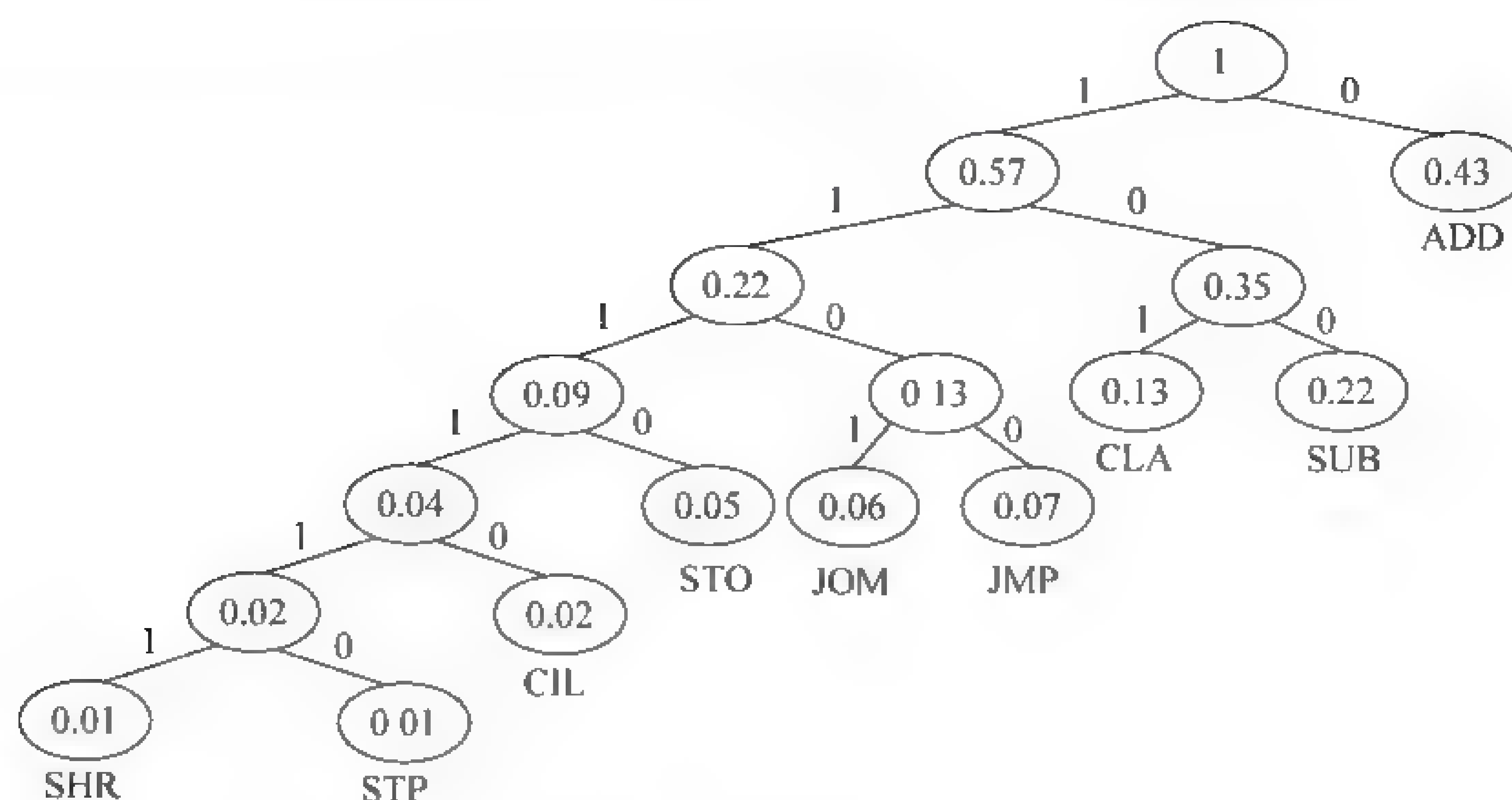


图 2.3 哈夫曼树(2)

9 条指令操作码如表 2.5 所示。

表 2.5 9 条指令操作码

指　　令	使用频度	哈夫曼编码 1	哈夫曼编码 2	3/3/3 扩展编码	2/7 扩展编码
ADD	0.43	0	0	0 0	0 0
CLA	0.22	1 0	1 0 0	0 1	0 1
SUB	0.13	1 1 0	1 0 1	1 0	1 0 0 0
JMP	0.07	1 1 1 0 0	1 1 0 0	1 1 0 0	1 0 0 1
JOM	0.06	1 1 1 0 1	1 1 0 1	1 1 0 1	1 0 1 0
STO	0.05	1 1 1 1 0	1 1 1 0	1 1 1 0	1 0 1 1
CIL	0.02	1 1 1 1 1 0	1 1 1 1 0	1 1 1 1 0 0	1 1 0 0
SHR	0.01	1 1 1 1 1 1 0	1 1 1 1 1 0	1 1 1 1 0 1	1 1 0 1
STP	0.01	1 1 1 1 1 1 1	1 1 1 1 1 1	1 1 1 1 1 0	1 1 1 0

哈夫曼编码的平均码长：2.42 位。3/3/3 扩展编码的平均码长：2.52 位。2/7 扩展编码的平均码长为：2.70 位。

【题 2.36】

解：根据题意，两地址指令格式如图 2.4 所示。

其中，4 位操作码可表示 16 个(2^4)短操作码。两地址指令共有 A 条，占用了 16 个短码点中的 A 个，剩余的($16-A$)个码点均可用作扩展标志。

单地址指令格式如图 2.5 所示。



图 2.4 两地址指令格式

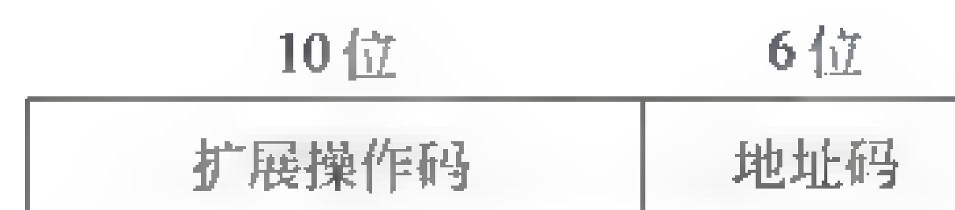


图 2.5 单地址指令格式

即每一个扩展标志都可使用一个 6 位的地址字段(两地址指令中的地址码 1)进行扩展，从而得到 2^6 个扩展操作码。所以，单地址指令最多可有$(16-A)\times 2^6$条。

【题 2.37】

解： 三地址指令的格式如图 2.6 所示。



图 2.6　三地址指令格式

操作码为 3 位长，可表示 $8(2^3)$ 个码点，用 4 个码点表示 4 条三地址指令，剩下的 4 个码点作为扩展标志。

单地址指令只需用地址码 3 字段表示地址，其余 6 位地址码可用于表示操作码。6 位长的扩展部分可表示 $64(2^6)$ 个码点，有 4 个 3 位长的扩展标志，所以，共可表示 9 位长的码点 $4\times2^6=256$ 个。若用 255 个码点表示 255 条单地址指令操作码，则还余下一个码点作为扩展标志。

零地址指令不需要地址码，地址码 3 的 3 位可用于表示零地址指令的操作码。3 位长的扩展部分可表示 $8(2^3)$ 个码点。但是，9 位长的码点只剩下一个作为扩展标志，因此，只能表示 8 条零地址指令操作码，不能满足题目中的数量要求。

如果单地址指令为 254 条，则 9 位长的码点余下 2 个码点作为扩展标志，再扩展 3 位后正好表示 16 条零地址指令操作码。

【题 2.38】

解：(1)哈夫曼树如图 2.7 所示。

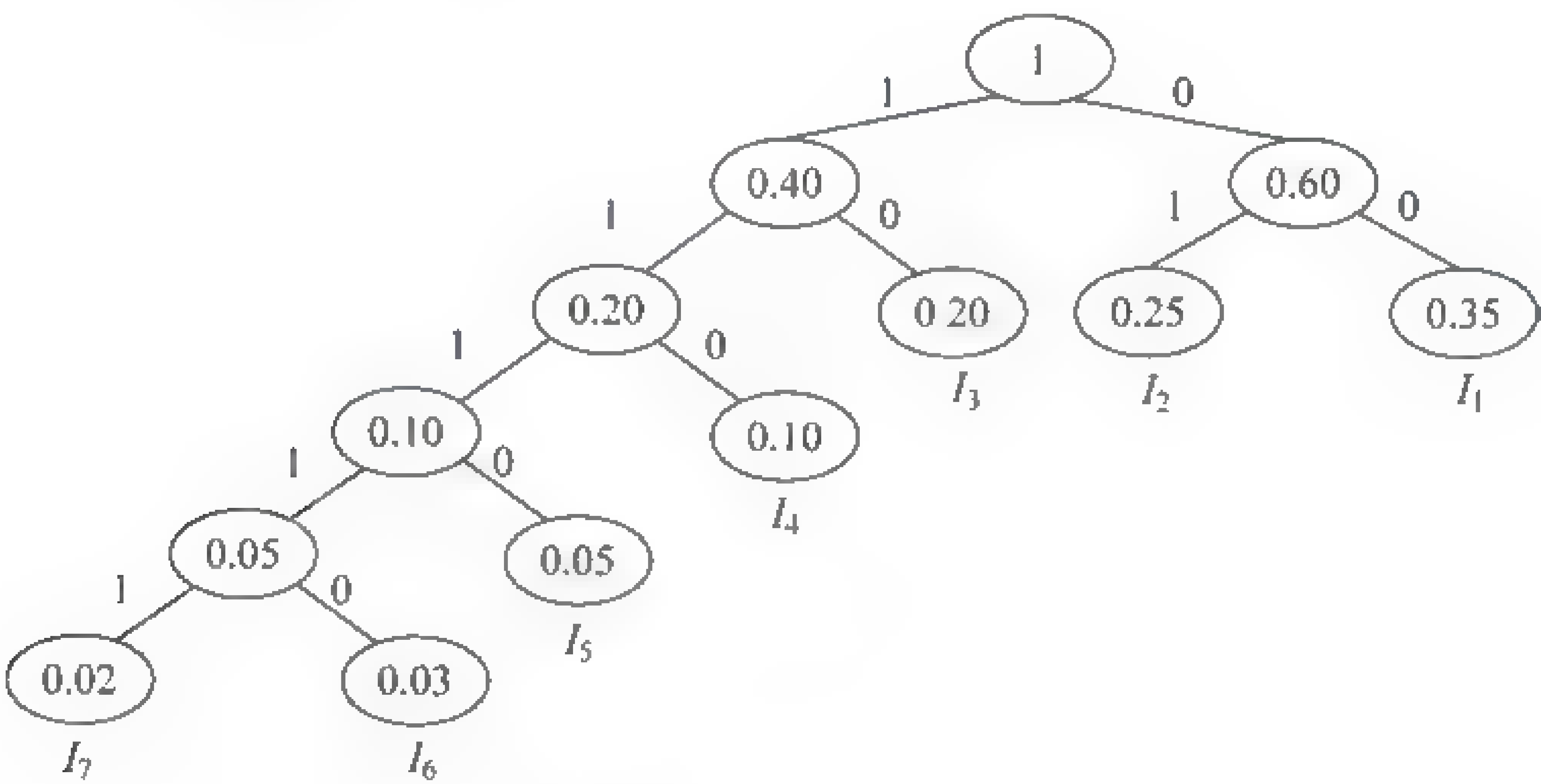


图 2.7　哈夫曼树

哈夫曼编码如表 2.6 所示。

表 2.6　哈夫曼编码

指　　令	使用频度 p_i	哈夫曼编码	3/4 扩展编码
I_1	0.35	0 0	0 0
I_2	0.25	0 1	0 1
I_3	0.20	1 0	1 0
I_4	0.10	1 1 0	1 1 0 0

续表

指　　令	使用频度 p_i	哈夫曼编码	3/4 扩展编码
I_5	0.05	1 1 1 0	1 1 0 1
I_6	0.03	1 1 1 1 0	1 1 1 0
I_7	0.02	1 1 1 1 1	1 1 1 1

7 条指令的操作码平均码长：$L = \sum p_i l_i = 2.35$(位)

(2) 3 条 8 位长的寄存器-寄存器型指令的格式如图 2.8 所示。

其中，8 个通用寄存器需用 3 位寄存器号来编址。2 位操作码字段表示 3 条指令操作码，一个码点作为扩展标志。

4 条 16 位长的寄存器-存储器型变址寻址指令的格式如图 2.9 所示。

2 位	3 位	3 位
操作码	寄存器号1	寄存器号2

图 2.8　寄存器-寄存器型指令的格式

4位	3位	1位	8位
操作码	寄存器号	变址寄存器号	偏移量

图 2.9　寄存器-存储器型变址寻址指令的格式

其中，两个变址寄存器用 1 位变址寄存器号来编址，8 位偏移量表示变址范围 −127～+127。4 位操作码中，扩展标志占 2 位，另外 2 位正好表示 1 条指令的操作码。

这样得到 3/4 扩展编码如表 2.6 所示。

【题 2.39】

解：(1) 二地址指令中，用 12 位表示 2 个 6 位的地址码，操作码字段为 4 位，可表示 16 个码点，其中 15 个(0000～1110)用来表示 15 条二地址指令操作码，一个码点 1111 作为扩展标志。

要求单地址指令和零地址指令的条数基本相等，那么，可以采用 15/63/64 扩展编码，即单地址指令 63 条，零地址指令 64 条。编码如图 2.10 所示。

(2) 要求 3 类指令条数的比例为 1∶9∶9，那么 3 类指令条数可以分别为 14 条、126 条和 128 条。编码如图 2.11 所示。

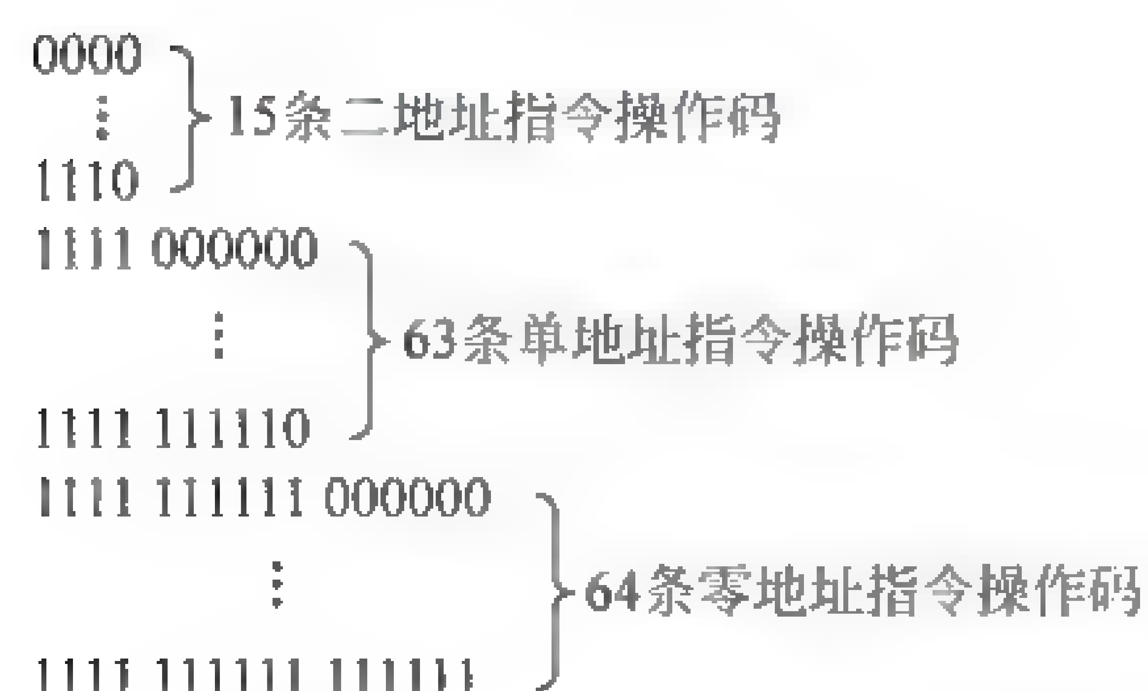


图 2.10　15/63/64 扩展编码

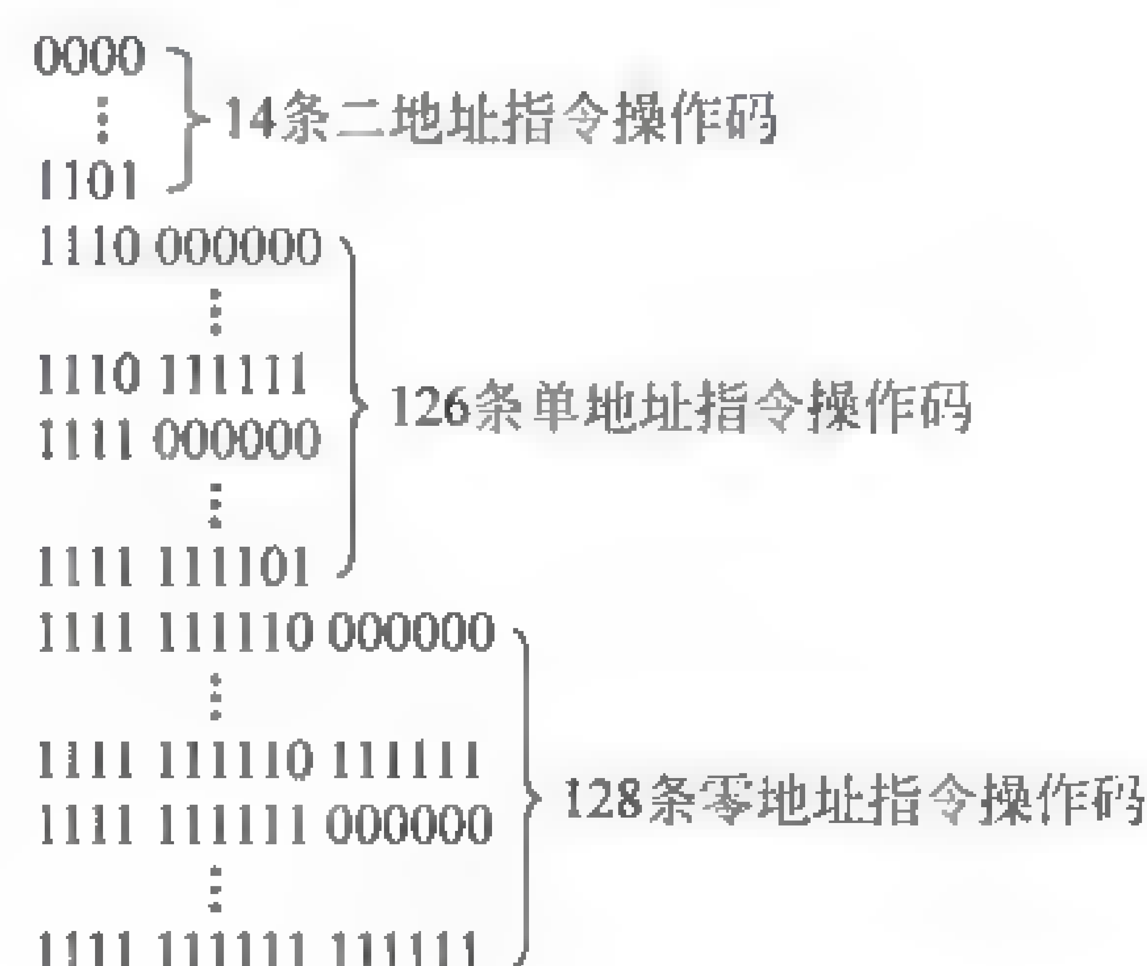


图 2.11　指令操作码

【题 2.40】

解：哈夫曼树如图 2.12 所示。

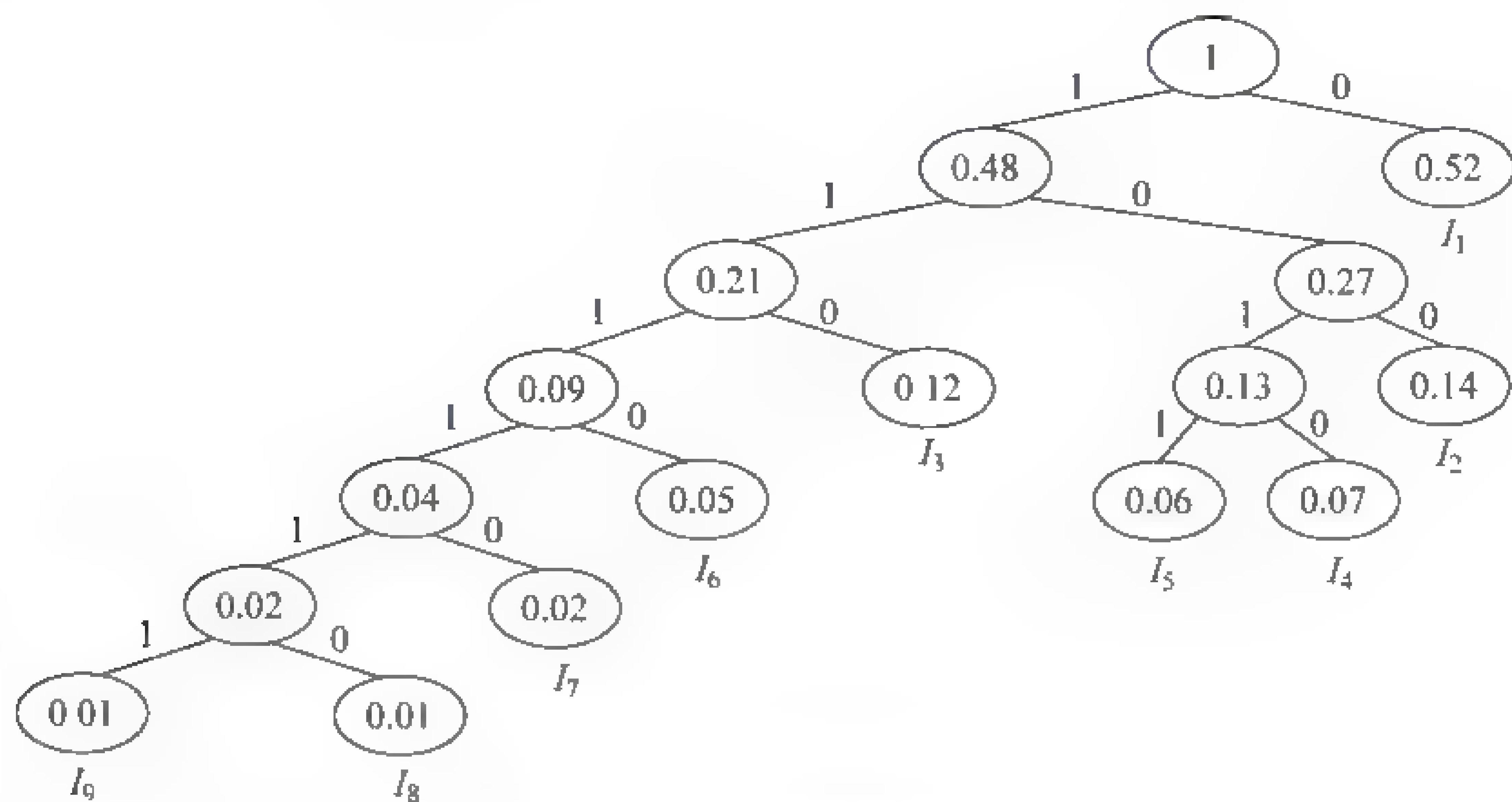


图 2.12 哈夫曼树

由哈夫曼树生成的哈夫曼编码和 2-4-6 扩展编码如表 2.7 所示。

表 2.7 哈夫曼编码和 2-4-6 扩展编码

指令	使用频度 p_i	哈夫曼编码	2-4-6 编码
I_1	0.52	0	0 0
I_2	0.14	1 0 0	0 1
I_3	0.12	1 1 0	1 0
I_4	0.07	1 0 1 0	1 1 0 0
I_5	0.06	1 0 1 1	1 1 0 1
I_6	0.05	1 1 1 0	1 1 1 0
I_7	0.02	1 1 1 1 0	1 1 1 1 0 0
I_8	0.01	1 1 1 1 1 0	1 1 1 1 0 1
I_9	0.01	1 1 1 1 1 1	1 1 1 1 1 0

哈夫曼编码的平均码长：$L_1 = \sum_{i=1}^{9} p_i l_i = 2.06$(位)

2-4-6 扩展编码的平均码长：$L_2 = \sum_{i=1}^{9} p_i l_i = 2.52$(位)

【题 2.41】

解：

(1) $H = -\sum_{i=1}^{7} p_i \log_2 p_i = 2.17$

(2) 其哈夫曼树如图 2.13 所示，该树的每个叶结点分别对应于一条指令。在该树中，对每个结点向下的两个分支，分别用二进制“1”和“0”来表示。

从该哈夫曼树可以很容易地写出哈夫曼编码。表 2.8 中列出了所有指令的哈夫曼编码。

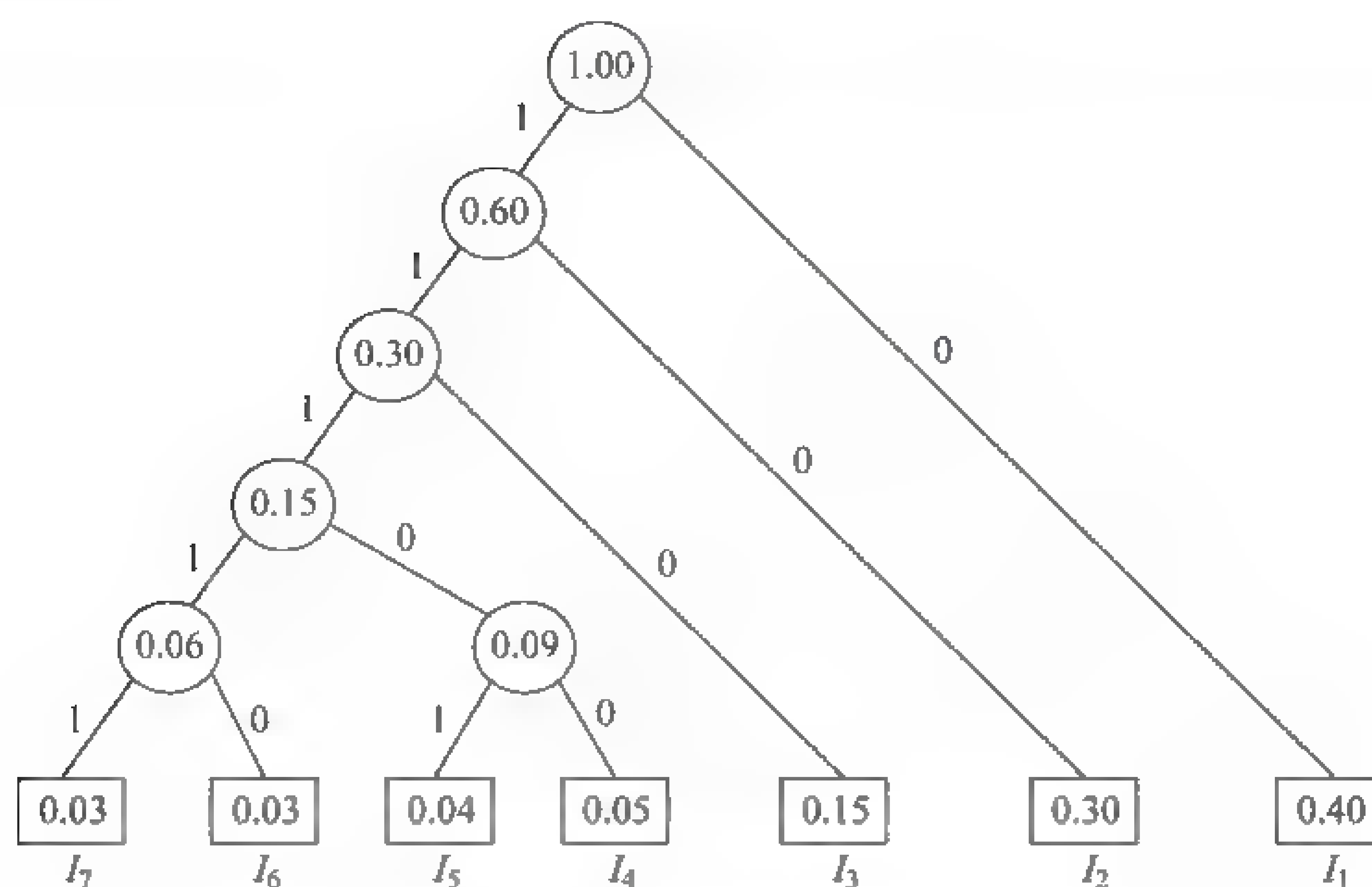


图 2.13　哈夫曼树举例

表 2.8　操作码的哈夫曼编码

指令	频度 p_i	操作码使用哈夫曼编码	操作码长度 l_i
I_1	0.40	0	1
I_2	0.30	1 0	2
I_3	0.15	1 1 0	3
I_4	0.05	1 1 1 0 0	5
I_5	0.04	1 1 1 0 1	5
I_6	0.03	1 1 1 1 0	5
I_7	0.03	1 1 1 1 1	5

该哈夫曼编码的平均码长是：

$$L = \sum_{i=1}^{7} p_i l_i = 2.20$$

其信息冗余量为：

$$\frac{2.20 - 2.17}{2.20} \approx 1.36\%$$

第 3 章　流水线技术

3.1　基本要求与难点

3.1.1　基本要求

(1) 掌握有关流水线的基本概念。

(2) 掌握流水线的工作原理，了解如何从不同的角度对流水线进行分类。

(3) 理解流水线的各项性能指标，能熟练地用时空图或公式计算吞吐率、加速比和效率。能熟练地画出时空图。

(4) 掌握消除流水线瓶颈的方法。

(5) 熟练掌握单功能非线性流水线的最优调度方法。能熟练地画出状态转换图，并根据状态转换图写出最优调度方案。了解双功能非线性流水线的最优调度方法。

(6) 掌握经典 5 段流水线的结构。

(7) 理解 3 种相关和 3 种冲突的概念，掌握解决结构冲突、数据冲突和控制冲突的方法，特别是：如何利用定向技术来解决数据冲突？如何用猜测法和延迟分支来解决控制冲突？

(8) 掌握基本的 MIPS 流水线的组成以及在各流水段完成的操作。掌握对该流水线进行改进后(把分支延迟减少为一个时钟周期)IF 段和 ID 段的操作的变化。

3.1.2　难点

(1) 流水线时空图的画法，如何计算流水线的吞吐率、加速比和效率？

(2) 单功能非线性流水线的最优调度方法。

(3) 如何解决结构冲突、数据冲突和控制冲突？

(4) 基本的 MIPS 流水线的各段完成的操作。

3.2　知识要点

3.2.1　流水线的基本概念

1. 什么是流水线

在计算机中，把一个重复的过程分解为若干个子过程，每个子过程由专门的功能部件来

实现。把多个处理过程在时间上错开，依次通过各功能段，这样，每个子过程就可以与其他的子过程并行进行。这就是**流水线技术**。流水线中的每个子过程及其功能部件称为流水线的级或段，流水线的段数称为**流水线的深度**。

一般采用时空图来描述流水线的工作过程。时空图的横坐标表示时间，纵坐标表示空间，即流水线的段。

流水线技术具有以下特点。

(1) 流水线实际上是把一个大的处理功能部件分解为多个独立的功能部件，并依靠它们的并行工作来提高吞吐率。

(2) 流水线中各段的时间应尽可能相等，否则将引起流水线堵塞和断流。因为时间最长的段将成为**流水线的瓶颈**。

(3) 流水线每个段的后面都要有一个缓冲寄存器(锁存器)，称为**流水寄存器**。

(4) 流水技术适合于大量重复的时序过程。

(5) 流水线需要有通过时间和排空时间。它们分别是指第一个任务和最后一个任务从进入流水线到流出结果的那个时间段。在这两个时间段中，流水线都不是满负荷工作。

2. 流水线的分类

流水线可以从不同的角度和观点来分类。下面是几种常见的分类。

1) 部件级、处理机级及系统级流水线

这是按照流水技术用于计算机系统的等级不同来分的。

部件级流水线是把处理机中的部件进行分段，再把这些部件分段相互连接而成。它使得运算操作能够按流水方式进行。这种流水线也称为**运算操作流水线**。

处理机级流水线又称**指令流水线**。它是把指令的执行过程按照流水方式进行处理，即把一条指令的执行过程分解为若干个子过程，每个子过程在独立的功能部件中执行。

系统级流水线是把多个处理机串行连接起来，对同一数据流进行处理，每个处理机完成整个任务中的一部分。前一个处理机的输出结果存入存储器中，作为后一个处理机的输入。这种流水线又称为**宏流水线**。

2) 单功能流水线与多功能流水线

单功能流水线是指流水线的各段之间的连接固定不变、只能完成一种固定功能的流水线。**多功能流水线**是指各段可以进行不同的连接，以实现不同功能的流水线。

3) 静态流水线与动态流水线

静态流水线是指在同一时间内，多功能流水线中的各段只能按同一种功能的连接方式工作的流水线。当流水线要切换到另一种功能时，必须等前面的任务都流出流水线之后，才能改变连接。

动态流水线是指在同一时间内，多功能流水线中的各段可以按照不同的方式连接，同时执行多种功能的流水线。

一般来说，动态流水线的效率比静态流水线的高。但是，动态流水线的控制要复杂得多。

4) 线性流水线与非线性流水线

线性流水线是指各段串行连接、没有反馈回路的流水线。数据通过流水线中的各段时，

每一个段最多只流过一次。**非线性流水线**是指各段除了有串行的连接外，还有反馈回路的流水线。在非线性流水线中，一个重要的问题是确定什么时候向流水线引进新的任务，才能使该任务不会与流水线中的任务发生争用流水段的冲突。这就是**非线性流水线的调度问题**。

5）顺序流水线与乱序流水线

这是根据流水线中任务流入和流出的顺序是否相同来分的。在**顺序流水线**中，流水线输出端任务流出的顺序与输入端任务流入的顺序完全相同。而在**乱序流水线**中，流水线输出端任务流出的顺序与输入端任务流入的顺序可以不同，允许后进入流水线的任务先完成。这种流水线又称为无序流水线或错序流水线。

通常把指令执行部件中采用了流水线的处理机称为流水线处理机。如果处理机具有向量数据表示和向量指令，则称之为**向量流水处理机**，简称向量机；否则就称之为**标量流水处理机**。

3.2.2　流水线的性能指标

衡量流水线性能的主要指标有吞吐率、加速比和效率。

1. 流水线的吞吐率

流水线的吞吐率TP(Through Put)是指在单位时间内流水线所完成的任务数量或输出结果的数量。

$$TP = \frac{n}{T_k} \tag{3.1}$$

其中，n为任务数，T_k是处理完成n个任务所用的时间。

1）各段时间均相等的流水线

$$TP = \frac{n}{(k+n-1)\Delta t} \tag{3.2}$$

其中，Δt为各段的时间，k为段数。

2）各段时间不完全相等的流水线

各段时间不等的流水线的实际吞吐率为：

$$TP = \frac{n}{\sum_{i=1}^{k}\Delta t_i + (n-1)\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)} \tag{3.3}$$

其中，Δt_i为第i段的时间，共有k个段。分母中的第一部分是流水线完成第一个任务所用的时间，第二部分是完成其余$n-1$个任务所用的时间。

流水线的最大吞吐率为：

$$TP_{max} = \frac{1}{\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)} \tag{3.4}$$

从式(3.3)和式(3.4)可以看出，当流水线各段的时间不完全相等时，流水线的最大吞吐率和实际吞吐率由时间最长的那个段决定，这个段就成了整条流水线的瓶颈。

可以用以下两种方法来消除瓶颈段。

(1) 细分瓶颈段。这是把流水线中的瓶颈段切分为几个独立的功能段，从而使流水线各段的处理时间都相等。

(2) 重复设置瓶颈段。如果无法把瓶颈段再细分，则可以采用重复设置瓶颈段的方法来解决问题。重复设置的段并行工作，在时间上依次错开处理任务。

2. 流水线的加速比

流水线的加速比是指使用顺序处理方式处理一批任务所用的时间（设为 T_s）与按流水处理方式处理同一批任务所用的时间（设为 T_k）之比：

$$S=\frac{T_s}{T_k} \tag{3.5}$$

假设流水线各段时间相等，都是 Δt，则该流水线的实际加速比为：

$$S=\frac{nk}{k+n-1} \tag{3.6}$$

当流水线的各段时间不完全相等时，一条 k 段流水线完成 n 个连续任务的实际加速比为：

$$S=\frac{n\sum_{i=1}^{k}\Delta t_i}{\sum_{i=1}^{k}\Delta t_i+(n-1)\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)} \tag{3.7}$$

3. 流水线的效率

流水线的效率即流水线设备的利用率，它是指流水线中的设备实际使用时间与整个运行时间的比值。如果各段时间相等，则各段的效率是相同的，且都等于整条流水线的效率。

$$E=\frac{n}{k+n-1}$$

在各段时间不等的情况下，k 段流水线连续处理 n 个任务的流水线效率为：

$$E=\frac{n\cdot\sum_{i=1}^{k}\Delta t_i}{k\left[\sum_{i=1}^{k}\Delta t_i+(n-1)\cdot\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)\right]} \tag{3.8}$$

计算流水线效率的一般公式可以表示为：

$$E=\frac{n\text{ 个任务实际占用的时空区的面积}}{k\text{ 个段总的时空区的面积}} \tag{3.9}$$

画出流水线的时空图，然后根据式(3.9)来计算效率，是一种比较直观通用的方法。对于线性流水线、非线性流水线、多功能流水线、任务不连续的情况等都适用。

4. 流水线设计中的若干问题

1) 瓶颈问题

当流水线各段时间不相等时，时间最大的那个段就成了瓶颈。机器的时钟周期取决于这个瓶颈段的延迟时间。因此，在设计流水线时，要尽可能使各段时间相等。

2）流水线的额外开销

流水线的额外开销由两部分构成：流水寄存器延迟和时钟偏移开销。增加流水线的段数可以提高流水线的性能，但是流水线段数的增加是受到这些额外开销限制的。

3）冲突问题

如果流水线中的指令或数据之间存在关联，则它们可能要相互等待，引起访问冲突，造成流水线的停顿。如何处理好冲突问题，是流水线设计中要解决的重要问题之一。

3.2.3 非线性流水线的调度

1. 单功能非线性流水线的最优调度

向一条非线性流水线的输入端连续输入两个任务之间的时间间隔称为非线性流水线的**启动距离**。而会引起非线性流水线功能段使用冲突的启动距离则称为**禁用启动距离**。启动距离和禁用启动距离一般都用时钟周期数来表示，是一个正整数。

对流水线的任务进行优化调度和控制的步骤如下。

(1) 根据预约表写出禁止表 F。

(2) 根据禁止表 F 写出初始冲突向量 $\boldsymbol{C}_0=(c_N c_{N-1}\cdots c_i\cdots c_2 c_1)$。

(3) 根据初始冲突向量 $\boldsymbol{C}_0$ 画出状态转换图。

令 $\boldsymbol{C}_k=\boldsymbol{C}_0$，按下式计算新的冲突向量：

$$\mathrm{SHR}^{(j)}(\boldsymbol{C}_k)\vee \boldsymbol{C}_0 \qquad (3.10)$$

其中，$\mathrm{SHR}^{(j)}$ 表示逻辑右移 j 位。对于所有允许的时间间隔都按上述步骤求出其新的冲突向量，并且把新的冲突向量作为当前冲突向量，反复使用上述步骤，直到不再产生不同的冲突向量为止。由此可以画出用冲突向量表示的**流水线状态转移图**。

(4) 根据状态转换图写出最优调度方案。

由初始状态出发，任何一个闭合回路即为一种调度方案。为了找到最佳的调度方案，只要列出所有可能的调度方案，计算出每种方案的平均时间间隔，从中找出其最小者即可。

2. 多功能非线性流水线的调度

双功能(设为功能 A 和 B)非线性流水线的最优调度方法类似于单功能非线性流水线的调度方法。只是其状态转移图中结点状态的表示不同，是由两个冲突向量构成的冲突矩阵。其初始结点有两个，分别对应于第一个任务是 A 类和 B 类的情况。

3.2.4 流水线的相关与冲突

1. 一个经典的 5 段流水线

先考虑在非流水情况下是如何实现的。我们把一条指令的执行过程分为以下 5 个时钟周期。

1）取指令周期(IF)

以程序计数器 PC 中的内容作为地址，从存储器中取出指令并放入指令寄存器 IR；同

时PC值加4(假设每条指令占4个字节),指向顺序的下一条指令。

2) 指令译码/读寄存器周期(ID)

对指令进行译码,并用IR中的寄存器地址去访问通用寄存器组,读出所需的操作数。

3) 执行/有效地址计算周期(EX)

(1) load和store指令：ALU把指定寄存器的内容与偏移量相加,形成访存有效地址。

(2) ALU指令：ALU对从通用寄存器组中读出的数据进行运算。

(3) 分支指令：ALU把偏移量与PC值相加,形成转移目标的地址。同时,判断分支是否成功。

4) 存储器访问/分支完成周期(MEM)

(1) load和store指令：根据有效地址从存储器中读出相应的数据(load指令)；或者是把指定的数据写入有效地址所指出的存储器单元(store指令)。

(2) 分支指令：如果分支"成功",就把在前一个周期中计算好的转移目标地址送入PC。分支指令执行完成。否则,就不进行任何操作。

5) 写回周期(WB)

把结果写入通用寄存器组。对于ALU运算指令来说,这个结果来自ALU,而对于load指令来说,这个结果是来自存储器。

把上述实现方案改造为流水线实现是比较简单的,只要把上面的每一个周期作为一个流水段,并在各段之间加上锁存器,就构成了一个经典的5段流水线,如图3.1所示。这些锁存器称为流水线寄存器。如果在每个时钟周期启动一条指令,则采用流水方式后的性能将是非流水方式的5倍。当然,事情也没这么简单,还要解决好流水处理带来的一些问题。

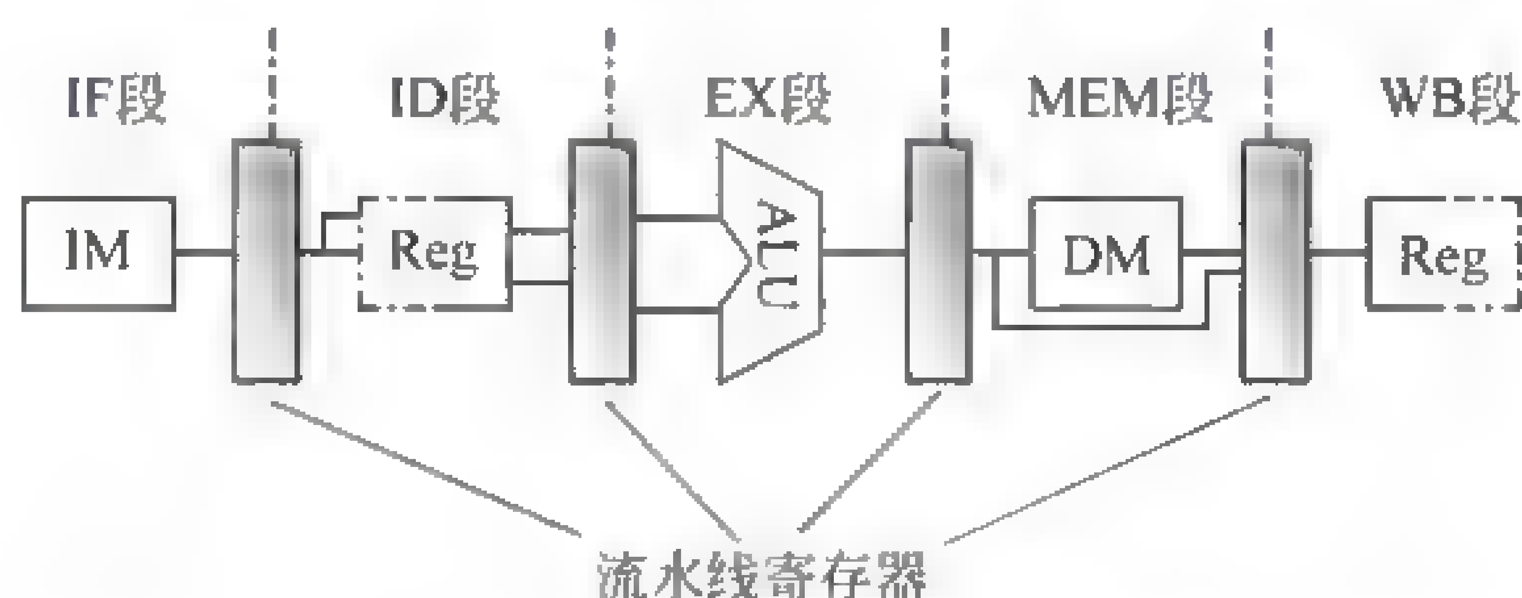


图3.1 一个经典的5段流水线

第一,要保证不会在同一时钟周期要求同一个功能段做两件不同的工作。

第二,为了避免IF段的访存(取指令)与MEM段的访存(读/写数据)发生冲突,必须采用分离的指令存储器和数据存储器,或者是仍采用一个公用的存储器,但要采用分离的指令Cache和数据Cache。一般是采用后者。

第三,ID段要对通用寄存器组进行读操作,而WB段要对通用寄存器组进行写操作,为了解决对同一通用寄存器的访问冲突,我们把写操作安排在时钟周期的前半拍完成,把读操作安排在后半拍完成。

第四,没有考虑PC的问题。在每个时钟周期中,都要在IF段把PC值加4,为此需要设置一个专门的加法器。另外,分支指令在MEM段也要修改PC的值。

2. 相关与流水线冲突

• 相关

相关是指两条指令之间存在某种依赖关系。如果两条指令相关,那么它们可能就不能在流水线中重叠执行或者只能部分重叠。

相关有3种类型:数据相关(也称真数据相关),名相关,控制相关。

1) 数据相关

考虑两条指令 i 和 j,i 在 j 的前面(下同),如果下述条件之一成立,则称指令 j 与指令 i 数据相关:

(1) 指令 j 使用指令 i 产生的结果;

(2) 指令 j 与指令 k 数据相关,而指令 k 又与指令 i 数据相关。

其中第二个条件表明,数据相关具有传递性。两条指令之间如果存在第一个条件所指出的相关的链,则它们是数据相关的。数据相关反映了数据的流动关系,即如何从其产生者流动到其消费者。

2) 名相关

这里的名是指指令所访问的寄存器或存储器单元的名称。如果两条指令使用了相同的名,但是它们之间并没有数据流动,则称这两条指令存在**名相关**。指令 j 与指令 i 之间的名相关有以下两种。

(1) 反相关。如果指令 j 所写的名与指令 i 所读的名相同,则称指令 i 和 j 发生了反相关。反相关指令之间的执行顺序是必须严格遵守的,以保证 i 读的值是正确的。

(2) 输出相关。如果指令 j 和指令 i 所写的名相同,则称指令 i 和 j 发生了输出相关。输出相关指令的执行顺序是不能颠倒的,以保证最后的结果是指令 j 写进去的。

与真数据相关不同,名相关的两条指令之间并没有数据的传送,只是使用了相同的名而已。可以通过改变指令中操作数的名来消除名相关,这就是**换名**技术。对于寄存器操作数进行换名称为**寄存器换名**。寄存器换名既可以用编译器静态实现,也可以用硬件动态完成。

3) 控制相关

控制相关是指由分支指令引起的相关。它需要根据分支指令的执行结果来确定后面该执行哪个分支上的指令。一般来说,为了保证程序应有的执行顺序,必须严格按照控制相关确定的顺序执行。

• 流水线冲突

流水线冲突是指对于具体的流水线来说,由于相关的存在,使得指令流中的下一条指令不能在指定的时钟周期开始执行。

流水线冲突有3种类型:结构冲突,数据冲突,控制冲突。

在后面的讨论中,我们约定:当一条指令被暂停时,在该暂停指令之后流出的所有指令都要被暂停,而在该暂停指令之前流出的指令则继续进行。显然,在整个暂停期间,流水线不会启动新的指令。

1) 结构冲突

在流水线处理机中,如果某种指令组合因为资源冲突而不能正常执行,则称该处理机有**结构冲突**。这种情况发生在功能部件不是完全流水或者资源份数不够时。

2）数据冲突

（1）数据冲突

当相关的指令彼此靠得足够近时，它们在流水线中的重叠执行或者重新排序会改变指令读/写操作数的顺序，使之不同于它们串行执行时的顺序。这就是发生了**数据冲突**。

按照指令读访问和写访问的先后顺序，可以将数据冲突分为3种类型。习惯上，这些冲突是按照流水线必须保持的访问顺序来命名的。考虑两条指令 i 和 j，且 i 在 j 之前进入流水线，可能发生的数据冲突有以下几种。

① 写后读冲突（Read After Write，RAW）。指令 j 用到指令 i 的计算结果，而且在 i 将结果写入寄存器之前就去读该寄存器，因而得到的是旧值。这是最常见的一种数据冲突，它对应于真数据相关。

② 写后写冲突（Write After Write，WAW）。指令 j 和指令 i 的结果寄存器相同，而且 j 在 i 写入之前就先对该寄存器进行了写入操作，从而导致写入顺序错误。最后在结果寄存器中留下的是 i 写入的值。这种冲突对应于输出相关。

写后写冲突仅发生在这样的流水线中：流水线中不只一个段可以进行写操作；或者指令被重新排序了（第5章介绍）。前面介绍的5段流水线不会发生写后写冲突。

③ 读后写冲突（Write After Read，WAR）。指令 j 的目的寄存器和指令 i 的源操作数寄存器相同，而且 j 在 i 读取该寄存器之前就先对它进行了写操作，导致 i 读到的值是错误的。这种冲突是由反相关引起的。

读后写冲突在前述5段流水线中不会发生。读后写冲突仅发生在这样的情况下：有些指令的写结果操作提前了，而且有些指令的读操作滞后了；或者指令被重新排序了。

（2）使用定向技术减少数据冲突引起的停顿

为了减少停顿时间，可以采用定向技术来解决写后读冲突。**定向技术**（也称为旁路）的关键思想是：在发生写后读相关的情况下，在计算结果尚未出来之前，后面等待使用该结果的指令并不见得是马上就要用该结果。如果能够将该计算结果从其产生的地方（ALU的出口）直接送到其他指令需要它的地方（ALU的入口），那么就可以避免停顿。

（3）需要停顿的数据冲突

并不是所有的数据冲突都可以用定向技术来解决。对于这种情况，为保证代码能在流水线中正确执行，需要设置一个称为“流水线互锁机制”的功能部件。一般来说，流水线互锁机制的作用是检测和发现数据冲突，并使流水线停顿，直至冲突消失。停顿是从等待相关数据的指令开始，到相应的指令产生该数据为止。停顿导致在流水线中插入气泡，使得被停顿指令的CPI增加了相应的时钟周期数。

（4）依靠编译器解决数据冲突

为了减少停顿，对于无法用定向技术解决的数据冲突，可以通过在编译时让编译器重新组织指令顺序来消除冲突，这种技术称为“指令调度”或“流水线调度”。实际上，对于各种冲突，都有可能用指令调度来解决。

3）控制冲突

在流水线中，控制冲突可能会比数据冲突造成更多的性能损失，所以同样需要得到很好的处理。执行分支指令的结果有两种，一种是分支“成功”，PC值改变为分支转移的目标地址。另一种则是“失败”，这时PC的值保持正常递增，指向顺序的下一条指令。对分支指令

"成功"的情况来说，是在条件判定和转移地址计算都完成后，才改变PC值。对于5段流水线来说，改变PC值是在MEM段进行的。

处理分支指令最简单的方法是"冻结"或者"排空"流水线。即一旦在流水线的译码段ID检测到分支指令，就暂停其后的所有指令的执行，直到分支指令到达MEM段、确定出是否成功并计算出新的PC值为止。然后，按照新的PC值取指令。在这种情况下，分支指令给流水线带来了三个时钟周期的延迟。显然，这种让流水线空等的方法不是一种好的选择。

由分支指令引起的延迟称为**分支延迟**。为减少分支延迟，可采取以下措施。

(1) 在流水线中尽早判断出(或者猜测)分支转移是否成功；

(2) 尽早计算出分支目标地址。

这两种措施要同时采用，缺一不可。因为只有判断出转移是否成功而且得到分支目标地址后才能进行转移。

在下面的讨论中，假设这两步工作被提前到ID段完成，即分支指令是在ID段的末尾执行完，所带来的分支延迟为一个时钟周期。

减少分支延迟的方法有许多种。下面只介绍3种通过软件(编译器)来处理的方法，这三种方法有一个共同的特点：它们对分支的处理方法在程序的执行过程中始终是不变的。它们要么总是预测分支成功，要么总是预测分支失败。

(1) 预测分支失败

当ID段检测到分支指令时，可以让流水线通过预测选择两条分支路径中的一条，继续处理后续指令。预测有两种选择：猜测分支成功，或者猜测分支失败。不管哪一种，都可以通过编译器来优化性能，即让代码中最常执行的路径与所选的预测方向一致。

预测分支失败的方法是沿失败的分支继续处理指令，就好像什么都没发生似的。当确定分支是失败时，说明预测正确，流水线正常流动；当确定分支是成功时，流水线就把在分支指令之后取出的指令转化为空操作，并按分支目标地址重新取指令执行。

采用这种方法处理分支指令的后续指令时，要保证分支结果出来之前不能改变处理机的状态，以便一旦猜错时，处理机能够回退到原先的状态。

(2) 预测分支成功

这种方法按分支成功的假设进行处理。当流水线ID段检测到分支指令后，一旦计算出了分支目标地址，就开始从该目标地址取指令执行。

(3) 延迟分支

这种方法的主要思想是从逻辑上"延长"分支指令的执行时间。把延迟分支看成是由原来的分支指令和若干个延迟槽构成。不管分支是否成功，都要按顺序执行延迟槽中的指令。在采用延迟分支的实际机器中，绝大多数的延迟槽都是一个，即：

分支指令

延迟槽

下面只讨论这种情况。

可以看出，只要分支延迟槽中的指令是有用的，流水线中就没有出现停顿，这时延迟分支的方法能很好地减少分支延迟。

放入延迟槽中的指令是由编译器来选择的。实际上延迟分支能否带来好处完全取决于

编译器能否把有用的指令调度到延迟槽中。这也是一种指令调度技术。常用的调度方法有3种：从前调度，从目标处调度，从失败处调度。

上述方法受到两个方面的限制，一个是可以被放入延迟槽中的指令要满足一定的条件，另一个是编译器预测分支转移方向的能力。为了提高编译器在延迟槽中放入有用指令的能力，许多处理机采用了分支取消机制。在这种机制中，分支指令隐含预测的分支执行方向。当分支的实际执行方向和事先所预测的一样时，执行延迟槽中的指令，否则就将该指令转化成空操作。

3.2.5　流水线的实现

1. MIPS的一种简单实现

考虑实现MIPS指令子集的一种简单数据通路。该数据通路的操作分成5个时钟周期：取指令，指令译码/读寄存器，执行/有效地址计算，存储器访问/分支完成，写回。下面只讨论整数指令的实现，包括数据字的load和store，等于0转移，整数ALU指令等。

在这个数据通路上，最多花5个时钟周期就能实现一条MIPS指令。这5个时钟周期及相应的操作如下。

1）取指令周期(IF)

```
IR←Mem[PC]
NPC←PC+4
```

2）指令译码/读寄存器周期(ID)

```
A←Regs[rs]
B←Regs[rt]
Imm←((IR16)^16 ## IR16..31)
```

这里准备的放在A、B和Imm中的数据在后面周期中也许用不上，但也没关系，并不影响程序执行的正确性。而且统一这样处理，可以减少硬件的复杂度，是有益无害的。

3）执行/有效地址计算周期(EX)

在这个周期，ALU对在前一个周期准备好的操作数进行运算。不同指令所进行的操作不同：

(1) load指令和store指令：ALUo←A+Imm

(2) 寄存器-寄存器ALU指令：ALUo←A funct B

funct字段给出了运算操作码。

(3) 寄存器-立即值ALU指令：ALUo←A op Imm

(4) 分支指令

```
ALUo←NPC+(Imm<<2);
cond←(A== 0)
```

这里只考虑一种分支，即BEQZ(Branch if EQual Zero)，它的判断操作是：判断是否为

0。判断的结果存入寄存器 cond，供以后使用。

这里将有效地址计算周期和执行周期合并为一个时钟周期，这是因为 MIPS 指令集采用 load/store 结构，任何指令都不会同时进行数据有效地址的计算、转移目标地址的计算和对数据进行运算。

4) 存储器访问/分支完成周期(MEM)

所有指令都要在该周期对 PC 进行更新。除了分支指令，其他指令都是做：PC←NPC。在该周期处理的指令只有 load、store 和分支 3 种指令。

(1) load 和 store 指令

load 指令：LMD←Mem[ALUo]

store 指令：Mem[ALUo]←B

(2) 分支指令

```
if(cond) PC ←ALUo   else   PC←NPC
```

5) 写回周期(WB)

(1) 寄存器-寄存器 ALU 指令：Regs[rd]← ALUo

(2) 寄存器-立即数 ALU 指令：Regs[rt]← ALUo

(3) load 指令：Regs[rt]←LMD

上述指令均是将结果写入通用寄存器组。

2. 基本的 MIPS 流水线

如果把上述实现方案中每一个时钟周期完成的工作看作是流水线的一段，就可以很容易将之改造为流水实现。流水段中的所有操作在一个时钟周期内完成，每个时钟周期启动一条新的指令。这里主要进行了以下改动。

(1) 设置了流水寄存器。

在段与段之间设置了流水寄存器。**流水寄存器**的名称用其相邻的两个段的名称来表示。流水寄存器中的字段用“x. y[s]”的形式来表示。其中，x 为流水寄存器名称，y 为具体子寄存器的名称，s 为字段名称。

流水寄存器的作用如下。

① 将各段的工作隔开，使得它们不会互相干扰；

② 保存相应段的处理结果；

③ 向后传递后面将要用到的数据或者控制信息。

(2) 增加了向后传递 IR 和从 MEM/WB. IR 回送到通用寄存器组的连接。后者用于实现结果回写到通用寄存器组。

(3) 将对 PC 的修改移到了 IF 段，以便 PC 能及时地加 4，为取下一条指令做好准备。

为了详细了解该流水线的工作情况，需要知道各种指令在每一个流水段进行什么样的操作，如表 3.1 所示。在 IF 段和 ID 段，所有指令的操作都一样。从 EX 段开始才区分不同的指令。表中 IR[rs]是指 IR 的第 6 位到第 10 位，即 $IR_{6..10}$；IR[rt]是指 $IR_{11..15}$；IR[rd]是指 $IR_{16..20}$。

表 3.1　MIPS 流水线的每个流水段的操作

<table>
<tr><th>流水段</th><th colspan="3">所 有 指 令</th></tr>
<tr><td>IF</td><td colspan="3">IF/ID. IR ←Mem[PC];
IF/ID. NPC,PC ←(if((EX/MEM. IR[op] == branch)& EX/MEM. cond)
{EX/MEM. ALUo} else {PC+4});</td></tr>
<tr><td>ID</td><td colspan="3">ID/EX. A ←Regs[IF/ID. IR[rs]]; ID/EX. B ←Regs[IF/ID. IR[rt]];
ID/EX. NPC ←IF/ID. NPC; ID/EX. IR ←IF/ID. IR;
ID/EX. Imm ←(IF/ID. IR_{16})^{16} # # IF/ID. $IR_{16..31}$;</td></tr>
<tr><td></td><td>ALU 指令</td><td>load/store 指令</td><td>分支指令</td></tr>
<tr><td>EX</td><td>EX/MEM. IR ←ID/EX. IR;
EX/MEM. ALUo ←
ID/EX. A funct ID/EX. B
或
EX/MEM. ALUo ←
ID/EX. A op ID/EX. Imm;</td><td>EX/MEM. IR ←ID/EX. IR;
EX/MEM. ALUo ←
ID/EX. A+ID/EX. Imm;
EX/MEM. B←ID/EX. B;</td><td>EX/MEM. IR ←ID/EX. IR;
EX/MEM. ALUo ←
ID/EX. NPC + ID/EX. Imm
<< 2;
EX/MEM. cond ←
(ID/EX. A==0);</td></tr>
<tr><td>MEM</td><td>MEM/WB. IR ←EX/MEM. IR;
MEM/WB. ALUo ←
EX/MEM. ALUo;</td><td>MEM/WB. IR ←EX/MEM. IR;
MEM/WB. LMD ←
Mem[EX/MEM. ALUo];
或
Mem[EX/MEM. ALUo] ←
EX/MEM. B;</td><td></td></tr>
<tr><td>WB</td><td>Regs[MEM/WB. IR[rd]] ←
MEM/WB. ALUo;
或
Regs[MEM/WB. IR[rt]] ←
MEM/WB. ALUo;</td><td>Regs[MEM/WB. IR[rt]] ←
MEM/WB. LMD;</td><td></td></tr>
</table>

为了使该流水线正常工作,还要解决好数据冲突的问题。对于该流水线而言,所有的数据冲突均可以在 ID 段检测到。如果存在数据冲突,就在相应的指令流出 ID 段之前将其暂停。完成该工作的硬件称为流水线的互锁机制。类似地,若采用了定向技术,我们可以在 ID 段确定需要什么样的定向,并设置相应的控制。按这样处理,就不必在流水过程中将已经改变了机器状态的指令挂起,可以降低流水线的硬件复杂度。另外一种处理方法是在使用操作数的那个时钟周期(上述流水线中的 EX 和 MEM 段)的开始检测冲突和确定必需的定向。

由于使用 load 的结果而引起的流水线互锁称为 **load 互锁**。在 load 已经进入 EX 段时,通过比较相应的寄存器,就能判断当前在 ID 段的指令是否由于使用那条 load 指令的结果而导致 RAW 冲突。如果存在,流水线互锁机制必须在流水线中插入停顿,并使当前正处于 IF 段和 ID 段的指令不再前进。为实现这点,只要将 ID/EX. IR 中的操作码改为全 0(全 0 表示空操作),并将 IF/ID 寄存器的内容回送到自己的入口。

定向逻辑要考虑的情况更多,因此其实现比上述冲突检测机制更复杂。类似地,它也是通过比较流水寄存器中的寄存器地址来确定的。

在该流水线中,分支指令的条件测试和分支目标地址计算是在 EX 段完成,对 PC 的修

改是在MEM段完成。它所带来的分支延迟是3个时钟周期。为了减少分支延迟,需尽早完成这些工作。如果只考虑BEQZ和BNEZ,就可以把这些工作提前到ID段进行。为此,需要在ID段增设一个加法器,用于计算分支目标地址,并把条件测试“ 0?”的逻辑电路移到ID段。这样,分支延迟就减少为1个时钟周期。改进后的流水线对分支指令的处理变成了如表3.2所示的操作。斜体部分表示与表3.1不同的操作。

表3.2　改进后流水线的分支操作

流水段	分支指令
IF	IF/ID.IR ←Mem[PC]; ***IF/ID.NPC,PC ←(if ((IF/ID[op] == branch) & (Regs[IF/ID.IR[rs]]== 0))*** ***{IF/ID.NPC+(IF/ID.IR$_{16}$)^{16} ## (IF/ID.IR$_{16..31}$<<2)} else {PC+4});***
ID	ID/EX.A ←Regs[IF/ID. IR[rs]]; ID/EX.B←Regs[IF/ID. IR[rt]]; ID/EX.IR ←IF/ID.IR; ID/EX.Imm ←(IF/ID.IR$_{16}$)^{16} ## IF/ID. IR$_{16..31}$;
EX	
MEM	
WB	

习　　题

1. 概念题

【题3.1】　解释下列名词

流水线技术	通过时间	排空时间	定向技术
部件级流水线	指令流水线	系统级流水线	单功能流水线
多功能流水线	静态流水线	动态流水线	线性流水线
非线性流水线	顺序流水线	乱序流水线	吞吐率
流水线的加速比	流水线的效率	相关	数据相关
名相关	反相关	输出相关	换名技术
控制相关	流水线冲突	结构冲突	数据冲突
控制冲突	写后读冲突	写后写冲突	读后写冲突

2. 选择题

【题3.2】　以下说法不正确的是(　　)。

A. 线性流水线是单功能流水线　　B. 动态流水线是多功能流水线

C. 静态流水线是多功能流水线　　D. 动态流水线只能是单功能流水线

【题3.3】　非线性流水线的特征是(　　)。

A. 一次运算中使用流水线中的多个段

B. 一次运算中要多次使用流水线中的某些功能段

C. 流水线中某些功能段在各次运算中的作用不同

D. 流水线的各功能段在不同运算中可以有不同的连接

【题3.4】　以下是某非线性流水线的调度方案：[(2,7)；(2,2,7)；(3,4)；(4)；(3,4,7)；(4,7)；(4,3)；(5)；(7)]。其中，平均延迟最小的等间隔调度方案是(　　)。

A. (4)　　B. (5)　　C. (3,4)　　D. (4,3)

【题3.5】　与线性流水线最大吞吐率有关的是(　　)。

A. 各个功能段的执行时间　　B. 最快的那一段的执行时间

C. 最慢的那一段的执行时间　　D. 最后功能段的执行时间

【题3.6】　在MIPS的指令流水线中，可能发生的冲突有(　　)。

A. 同一条指令的读操作与写操作之间的写后读冲突

B. 先流入的指令的写操作与后流入的指令的读操作之间的写后读冲突

C. 后流入的指令的写操作与先流入的指令的读操作之间的读后写冲突

D. 两条指令的写操作之间的写后写冲突

3. 填空题

【题3.7】　流水线中的每个子过程及其功能部件称为流水线的段，流水线的段数称为________。

【题3.8】　流水线中最慢的一段称为流水线的________。

【题3.9】　如果流水线处理机具有向量数据表示和向量指令，则称之为________流水处理机；否则就称之为________流水处理机。

【题3.10】　按照流水线所完成的功能来分，流水线可分为________和________。

【题3.11】　按照同一时间内各段之间的连接方式来分，流水线可分为________和________。

【题3.12】　按照流水的级别来分，流水线可分为________、________和________。

【题3.13】　按照流水线中是否有反馈回路来分，流水线可分为________和________。

【题3.14】　按照输出端任务流出顺序与输入端流入的任务顺序是否相同来分，流水线可分为________和________。

【题3.15】　有一条非线性流水线，其预约表为$F=\{2,4,5\}$，初始冲突向量为$C_0=(11010)$，则对于C_0，后续的两个冲突向量分别为________和________。

【题3.16】　流水线在连续流动达到稳定状态后所得到的吞吐率，称为________。

【题3.17】　消除流水线瓶颈的方法有________和________两种。

【题3.18】　相关有3种类型：________、________和________。

【题3.19】　指令之间的名相关有________和________两种。

【题3.20】　流水线冲突有________、________和________3种类型。

【题3.21】　按照指令读访问和写访问的先后顺序，可以将数据冲突分为________、________和________3种类型。

【题3.22】　由分支指令引起的延迟称为________。

【题3.23】　延迟分支方法有3种调度策略：________、________和________。

【题3.24】　基本的MIPS流水线分为5个段，分别是：________、________、________、

________和________。

4. 问答题

【题 3.25】 简述流水线技术的特点。

【题 3.26】 流水寄存器的作用是什么？

【题 3.27】 简述非线性流水线调度所要解决的问题。

【题 3.28】 解决流水线结构冲突的方法有哪些？

【题 3.29】 简述通过软件(编译器)来减少分支延迟的3种方法。这些方法的共同特点是什么？

【题 3.30】 简述延迟分支方法中的3种调度策略的优缺点。

5. 应用题

【题 3.31】 设一条指令的执行过程分成取指令、分析指令和执行指令三个阶段，每个阶段所需的时间分别为 Δt、Δt 和 $2\Delta t$ 。分别求出下列各种情况下，连续执行 N 条指令所需的时间。

(1) 顺序执行方式；

(2) 只有"取指令"与"执行指令"重叠；

(3) "取指令""分析指令"与"执行指令"重叠。

【题 3.32】 在一台单流水线多操作部件的处理机上执行下面的程序，取指令、指令译码各需一个时钟周期，MOVE、ADD 和 MUL 操作各需要 2 个、3 个和 4 个时钟周期。每个操作都在第一个时钟周期从通用寄存器中读操作数，在最后一个时钟周期把运算结果写到通用寄存器中。

```
K:     MOVE  R1,R0          ;R1←(R0)
K+1:   MUL   R0,R2,R1       ;R0←(R2)*(R1)
K+2:   ADD   R0,R3,R2       ;R0←(R3)+(R2)
```

画出指令执行的流水线时空图，并计算执行完 3 条指令共需要多少个时钟周期。

【题 3.33】 有一指令流水线如图 3.2 所示。

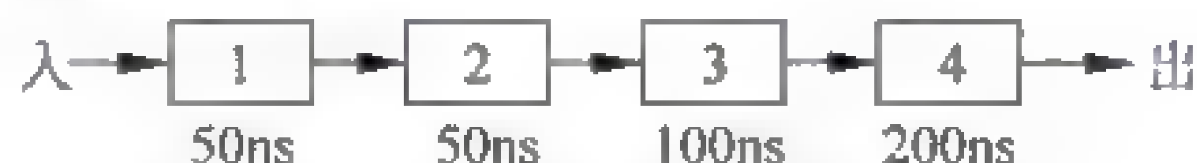


图 3.2　指令流水线

(1) 求连续输入 10 条指令，该流水线的实际吞吐率和效率；

(2) 该流水线的"瓶颈"在哪一段？请采取两种不同的措施消除此"瓶颈"。对于你所给出的两种新的流水线，连续输入 10 条指令时，其实际吞吐率和效率各是多少？

【题 3.34】 有一条流水线由 4 段组成，其中每当流经第 3 段时，总要在该段循环一次，然后才能流到第 4 段。如果每段经过一次所需要的时间都是 Δt，问：

(1) 当在流水线的输入端连续地每 Δt 时间输入任务时，该流水线会发生什么情况？

(2) 此流水线的最大吞吐率为多少？如果每 $2\Delta t$ 输入一个任务，连续处理 10 个任务时的实际吞吐率和效率是多少？

(3) 当每段时间不变时，如何提高该流水线的吞吐率？仍连续处理 10 个任务时，其吞吐率提高多少？

【题 3.35】 有一条静态多功能流水线由 5 段组成，如图 3.3 所示。加法用 1、3、4、5 段，乘法用 1、2、5 段，第 3 段的时间为 $2\Delta t$，其余各段的时间均为 Δt，而且流水线的输出可以直接返回输入端或暂存于相应的流水寄存器中。现要在该流水线上计算 $\prod_{i=1}^{4}(A_i+B_i)$，画出其时空图，并计算其吞吐率、加速比和效率。

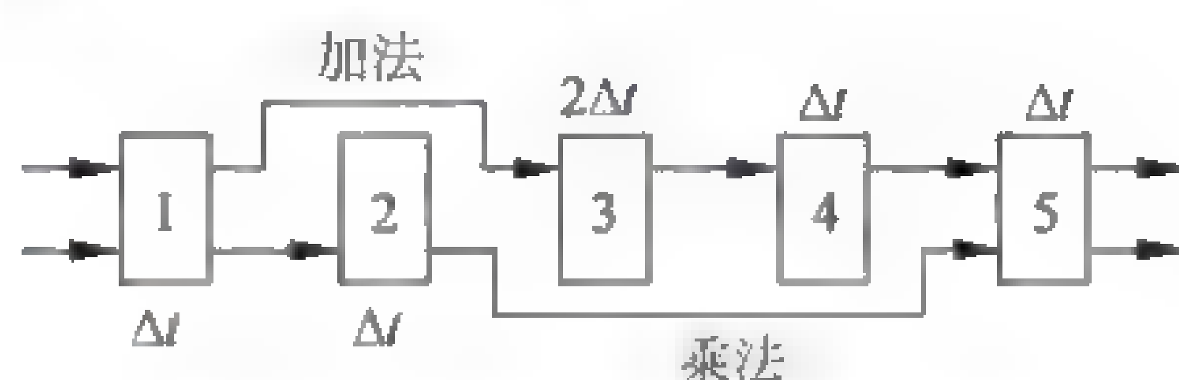


图 3.3　一条静态多功能流水线

【题 3.36】 有一条动态多功能流水线由 5 段组成(如图 3.4 所示)，加法用 1、3、4、5 段，乘法用 1、2、5 段，第 2 段的时间为 $2\Delta t$，其余各段时间均为 Δt，而且流水线的输出可以直接返回输入端或暂存于相应的流水寄存器中。若在该流水线上计算 $\sum_{i=1}^{4}(A_i\times B_i)$，试计算其吞吐率、加速比和效率。

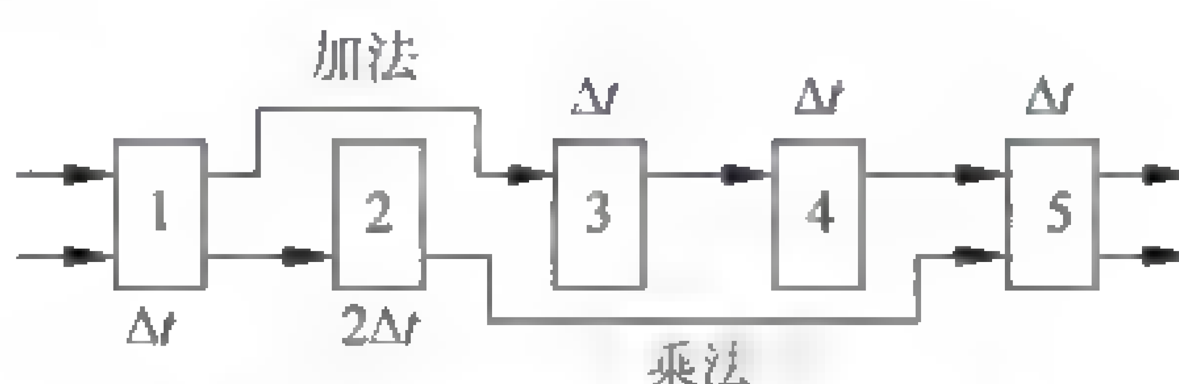


图 3.4　一条动态多功能流水线

【题 3.37】 有一条动态多功能流水线由 6 个功能段组成，如图 3.5 所示。

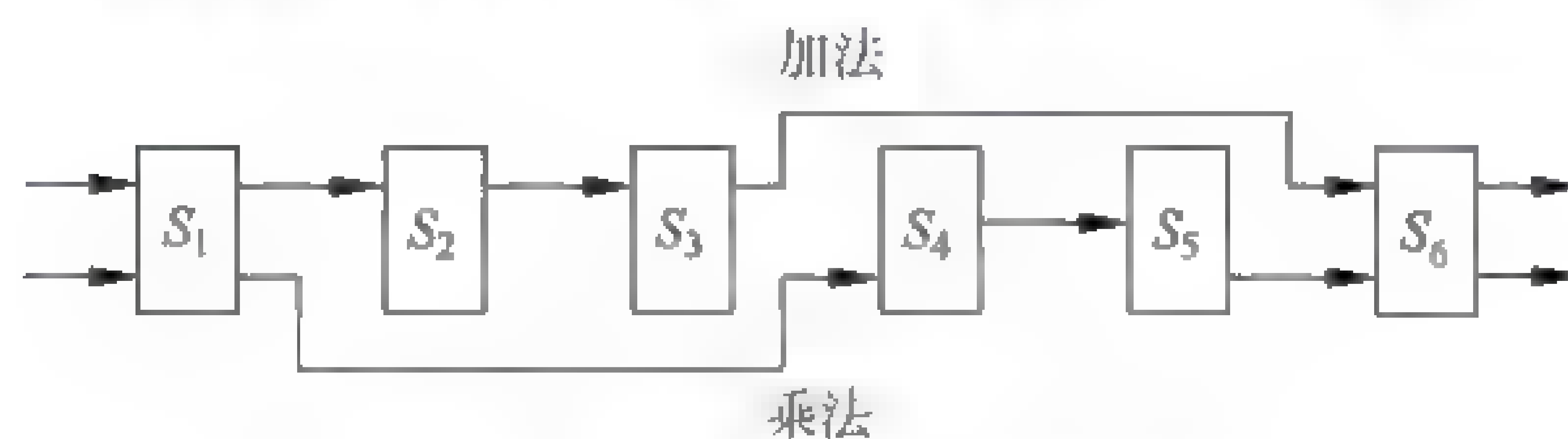


图 3.5　6 段动态多功能流水线

其中，S_1、S_4、S_5、S_6 组成乘法流水线，S_1、S_2、S_3、S_6 组成加法流水线，各个功能段时间均为 50ns，假设该流水线的输出结果可以直接返回输入端，而且设置有足够的缓冲寄存器，若以最快的方式用该流水线计算：

$$\sum_{i=1}^{5}x_i y_i z_i$$

(1) 画出时空图；

(2) 计算实际的吞吐率、加速比和效率。

【题 3.38】 有一条 4 段流水线如图 3.6 所示，一个任务通过此流水线的完成总时间为 $6\Delta t$，所有相继段必须在每个时钟周期之后才能使用。

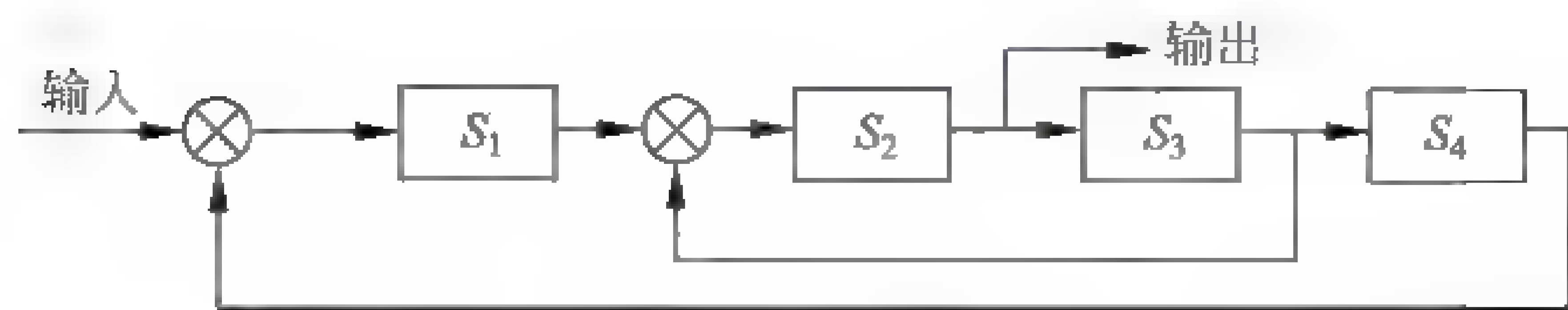


图3.6　一条4段流水线

写出该流水线的预约表(4行6列)。

【题3.39】 在一个5段流水线处理机上,各段执行时间均为 Δt,需经 $9\Delta t$ 才能完成一个任务,任务处理对各段的使用要求预约表如表3.3所示。

表3.3　题3.39预约表

功能段＼时间	1	2	3	4	5	6	7	8	9
S_1	√								√
S_2		√	√						
S_3				√			√	√	
S_4				√	√				
S_5						√	√		

(1) 画出流水线任务调度的状态转移图。

(2) 求出流水线的最优调度策略和流水线的最大吞吐率。

(3) 按最优调度策略连续输入6个任务,流水线的实际吞吐率是多少?

【题3.40】 有一个5段流水线,各段执行时间均为 Δt,其预约表如表3.4所示。

表3.4　题3.40预约表

功能段＼时间	1	2	3	4	5	6	7
S_1	√						√
S_2		√			√		
S_3			√	√			
S_4				√			√
S_5					√	√	

(1) 画出流水线任务调度的状态转移图。

(2) 分别求出允许不等时间间隔调度和等时间间隔调度的两种最优调度策略,以及这两种调度策略的流水线最大吞吐率。

(3) 若连续输入10个任务,求这两种调度策略的流水线实际吞吐率和加速比。

【题3.41】 在MIPS流水线上运行如下代码序列:

```
LOOP:   LW      R1,0(R2)
        DADDIU  R1,R1,#1
        SW      R1,0(R2)
        DADDIU  R2,R2,#4
        DSUB    R4,R3,R2
        BNEZ    R4,LOOP
```

其中,R3 的初值是 R2+396。假设:在整个代码序列的运行过程中,所有的存储器访问都是命中的,并且在一个时钟周期中对同一个寄存器的读操作和写操作可以通过寄存器组"定向"。问:

(1) 在没有任何其他定向(或旁路)硬件的支持下,请画出该指令序列执行的流水线时空图。假设采用排空流水线的策略处理分支指令,且所有的存储器访问都命中 Cache,那么执行上述循环需要多少个时钟周期?

(2) 假设该流水线有正常的定向路径,请画出该指令序列执行的流水线时空图。假设采用预测分支失败的策略处理分支指令,且所有的存储器访问都命中 Cache,那么执行上述循环需要多少个时钟周期?

(3) 假设该流水线有正常的定向路径和一个单周期延迟分支,请对该循环中的指令进行调度,你可以重新组织指令的顺序,也可以修改指令的操作数,但是注意不能增加指令的条数。请画出该指令序列执行的流水线时空图,并计算执行上述循环所需要的时钟周期数。

【题 3.42】 假设各种分支指令数占所有指令数的百分比如表 3.5 所示。

表 3.5 各种分支指令数占所有指令数的百分比

条件分支	20%(其中的 60%是分支成功的)
跳转和调用	5%

现有一条段数为 4 的流水线,无条件分支在第二个时钟周期结束时就被解析出来,而条件分支要到第 3 个时钟周期结束时才能够被解析出来。第一个流水段是完全独立于指令类型的,即所有类型的指令都必须经过第一个流水段的处理。请问在没有任何控制相关的情况下,该流水线相对于存在上述控制相关情况下的加速比是多少?

题 解

1. 概念题

【题 3.1】 解释下列名词

流水线技术——将一个重复的时序过程,分解成为若干个子过程,而每一个子过程都可有效地在其专用功能段上与其他子过程同时执行。

通过时间——流水线中第一个任务进入流水线到流出结果所需的时间。

排空时间——流水线中最后一个任务从进入流水线到流出结果所需的时间。

定向技术——用来解决写后读冲突。在发生写后读相关的情况下,在计算结果尚未出来之前,后面等待使用该结果的指令并不一定马上就要用该结果。如果能够将该计算结果从其产生的地方直接送到其他指令需要它的地方,那么就可以避免停顿。

部件级流水线——又称运算操作流水线。把处理机中的部件进行分段,再把这些部件分段相互连接而成。它使得运算操作能够按流水方式进行。

指令流水线——又称处理机级流水线。它是把指令的执行过程按照流水方式进行处理,即把一条指令的执行过程分解为若干个子过程,每个子过程在独立的功能部件中执行。

系统级流水线——又称为宏流水线。它是把多个处理机串行连接起来，对同一数据流进行处理，每个处理机完成整个任务中的一部分。前一个处理机的输出结果存入存储器中，作为后一个处理机的输入。

单功能流水线——指流水线的各段之间的连接固定不变、只能完成一种固定功能的流水线。

多功能流水线——指各段可以进行不同的连接，以实现不同功能的流水线。

静态流水线——指在同一时间内，多功能流水线中的各段只能按同一种功能的连接方式工作的流水线。当流水线要切换到另一种功能时，必须等前面的任务都流出流水线之后，才能改变连接。

动态流水线——指在同一时间内，多功能流水线中的各段可以按照不同的方式连接，同时执行多种功能的流水线。它允许在某些段正在实现某种运算时，另一些段却在实现另一种运算。

线性流水线——指各段串行连接、没有反馈回路的流水线。数据通过流水线中的各段时，每一个段最多只流过一次。

非线性流水线——指各段除了有串行的连接外，还有反馈回路的流水线。

顺序流水线——流水线输出端任务流出的顺序与输入端任务流入的顺序完全相同。

乱序流水线——流水线输出端任务流出的顺序与输入端任务流入的顺序可以不同，允许后进入流水线的任务先完成。这种流水线又称为无序流水线、错序流水线、异步流水线。

吞吐率——指在单位时间内流水线所完成的任务数量或输出结果的数量。

流水线的加速比——指使用顺序处理方式处理一批任务所用的时间与按流水处理方式处理同一批任务所用的时间之比。

流水线的效率——即流水线设备的利用率，它是指流水线中的设备实际使用时间与整个运行时间的比值。

相关——指两条指令之间存在某种依赖关系。

数据相关——考虑两条指令 i 和 j，i 在 j 的前面，如果下述条件之一成立，则称指令 j 与指令 i 数据相关：

(1) 指令 j 使用指令 i 产生的结果；

(2) 指令 j 与指令 k 数据相关，而指令 k 又与指令 i 数据相关。

名相关——如果两条指令使用了相同的名，但是它们之间并没有数据流动，则称这两条指令存在名相关。

反相关——考虑两条指令 i 和 j，i 在 j 的前面，如果指令 j 所写的名与指令 i 所读的名相同，则称指令 i 和 j 发生了反相关。

输出相关——考虑两条指令 i 和 j，i 在 j 的前面，如果指令 j 和指令 i 所写的名相同，则称指令 i 和 j 发生了输出相关。

换名技术——名相关的两条指令之间并没有数据的传送，只是使用了相同的名。可以把其中一条指令所使用的名换成别的，以消除名相关。

控制相关——指由分支指令引起的相关。它需要根据分支指令的执行结果来确定后面该执行哪个分支上的指令。

流水线冲突——指对于具体的流水线来说，由于相关的存在，使得指令流中的下一条指

令不能在指定的时钟周期开始执行。

结构冲突——因硬件资源满足不了指令重叠执行的要求而发生的冲突。

数据冲突——当指令在流水线中重叠执行时，因需要用到前面指令的执行结果而发生的冲突。

控制冲突——流水线遇到分支指令或其他会改变PC值的指令所引起的冲突。

写后读冲突——考虑两条指令 i 和 j，且 i 在 j 之前进入流水线，指令 j 用到指令 i 的计算结果，而且在 i 将结果写入寄存器之前就去读该寄存器，因而得到的是旧值。

写后写冲突——考虑两条指令 i 和 j，且 i 在 j 之前进入流水线，指令 j 和指令 i 的结果寄存器相同，而且 j 在 i 写入之前就先对该寄存器进行了写入操作，从而导致写入顺序错误。最后在结果寄存器中留下的是 i 写入的值。

读后写冲突——考虑两条指令 i 和 j，且 i 在 j 之前进入流水线，指令 j 的目的寄存器和指令 i 的源操作数寄存器相同，而且 j 在 i 读取该寄存器之前就先对它进行了写操作，导致 i 读到的值是错误的。

1. 选择题

【题3.2】 答：D
【题3.3】 答：B
【题3.4】 答：A
【题3.5】 答：C
【题3.6】 答：B

3. 填空题

【题3.7】 答：流水线的深度
【题3.8】 答：瓶颈
【题3.9】 答：向量、标量
【题3.10】 答：单功能流水线、多功能流水线
【题3.11】 答：静态流水线、动态流水线
【题3.12】 答：部件级流水线、处理机级流水线、处理机间流水线
【题3.13】 答：线性流水线、非线性流水线
【题3.14】 答：顺序流动流水线、异步流动流水线
【题3.15】 答：111110、11011
【题3.16】 答：最大吞吐率
【题3.17】 答：细分瓶颈段、重复设置瓶颈段
【题3.18】 答：数据相关、名相关、控制相关
【题3.19】 答：反相关、输出相关
【题3.20】 答：结构冲突、数据冲突、控制冲突
【题3.21】 答：写后读冲突、写后写冲突、读后写冲突
【题3.22】 答：分支延迟
【题3.23】 答：从前调度、从目标处调度、从失败处调度

【题3.24】 答：取指令周期、指令译码/读寄存器周期、执行/有效地址计算周期、存储器访问/分支完成周期、写回周期

4. 问答题

【题3.25】 答：流水技术具有以下特点。

(1) 流水线把一个处理过程分解为若干个子过程，每个子过程由一个专门的功能部件来实现。因此，流水线实际上是把一个大的处理功能部件分解为多个独立的功能部件，并依靠它们的并行工作来提高吞吐率。

(2) 流水线中各段的时间应尽可能相等，否则将引起流水线堵塞和断流。

(3) 流水线每一个功能部件的前面都要有一个缓冲寄存器，称为流水寄存器。

(4) 流水技术适合于大量重复的时序过程，只有在输入端不断地提供任务，才能充分发挥流水线的效率。

(5) 流水线需要有通过时间和排空时间。在这两个时间段中，流水线都不是满负荷工作。

【题3.26】 答：流水寄存器的作用是在相邻的两段之间传送信息，提供后面流水段要用到的信息。

【题3.27】 答：在非线性流水线中，由于反馈回路的存在，有可能会出现多个任务争用同一段的冲突现象。究竟应按什么样的时间间隔向流水线输入新任务，才能既不发生冲突，又能使流水线有较高的吞吐率和效率？这就是非线性流水线调度所要解决的问题。

【题3.28】 答：①流水化功能单元；②资源重复；③暂停流水线。

【题3.29】 答：(1)预测分支失败：沿失败的分支继续处理指令，就好像什么都没发生似的。当确定分支是失败时，说明预测正确，流水线正常流动；当确定分支是成功时，流水线就把在分支指令之后取出的指令转化为空操作，并按分支目标地址重新取指令执行。

(2) 预测分支成功：当流水线ID段检测到分支指令后，一旦计算出了分支目标地址，就开始从该目标地址取指令执行。

(3) 延迟分支：主要思想是从逻辑上"延长"分支指令的执行时间。把延迟分支看成是由原来的分支指令和若干个延迟槽构成。不管分支是否成功，都要按顺序执行延迟槽中的指令。

三种方法的共同特点：它们对分支的处理方法在程序的执行过程中始终是不变的。它们要么总是预测分支成功，要么总是预测分支失败。

【题3.30】 答：延迟分支方法中的3种调度策略的优缺点如表3.6所示。

表3.6　延迟分支方法中的3种调度策略的优缺点

调度策略	对调度的要求	对流水线性能改善的影响
从前调度	分支必须不依赖于被调度的指令	总是可以有效提高流水线性能
从目标处调度	如果分支转移失败，必须保证被调度的指令对程序的执行没有影响。可能需要复制被调度指令	分支转移成功时，可以提高流水线性能。但由于复制指令，可能加大程序空间
从失败处调度	如果分支转移成功，必须保证被调度的指令对程序的执行没有影响	分支转移失败时，可以提高流水线性能

5. 应用题

【题 3.31】

解：(1) 每条指令的执行时间为：$\Delta t + \Delta t + 2\Delta t = 4\Delta t$

连续执行 N 条指令所需的时间为：$4N\Delta t$

(2) 连续执行 N 条指令所需的时间为：$4\Delta t + 3(N-1)\Delta t = (3N+1)\Delta t$

(3) 连续执行 N 条指令所需的时间为：$4\Delta t + 2(N-1)\Delta t = (2N+2)\Delta t$

【题 3.32】

解：

指令/时钟	1	2	3	4	5	6	7	8	9
K	IF	ID	EX	EX					
$K+1$		IF	ID	stall	EX	EX	EX	EX	
$K+2$			IF	stall	ID	EX	EX	stall	EX

计算执行完三条指令共使用了 9 个时钟周期。

【题 3.33】

解：(1)

$$T_{\text{pipeline}} = \sum_{i=1}^{m} \Delta t_i + (n-1)\Delta t_{\max} = (50+50+100+200) + 9 \times 200 = 2200(\text{ns})$$

$$\text{TP} = \frac{n}{T_{\text{pipeline}}} = \frac{1}{220}(\text{ns}^{-1})$$

$$E = \text{TP} \cdot \frac{\sum_{i=1}^{m} \Delta t_i}{m} = \text{TP} \cdot \frac{400}{4} = \frac{5}{11} \approx 45.45\%$$

(2) 3、4 段是瓶颈。

① 变成 8 级流水线(细分)，如图 3.7 所示。

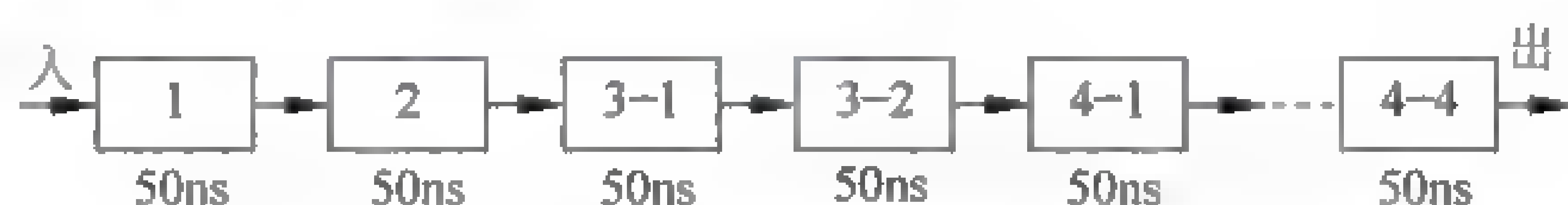


图 3.7　8 级流水线

$$T_{\text{pipeline}} = \sum_{i=1}^{m} \Delta t_i + (n-1)\Delta t_{\max} = 50 \times 8 + 9 \times 50 = 850(\text{ns})$$

$$\text{TP} = \frac{n}{T_{\text{pipeline}}} = \frac{1}{85}(\text{ns}^{-1})$$

$$E = \text{TP} \cdot \frac{\sum_{i=1}^{m} \Delta t_i}{m} = \text{TP} \cdot \frac{400}{8} = \frac{10}{17} \approx 58.82\%$$

② 重复设置部件(如图 3.8 所示)

$$\text{TP} = \frac{n}{T_{\text{pipeline}}} = \frac{1}{85}(\text{ns}^{-1})$$

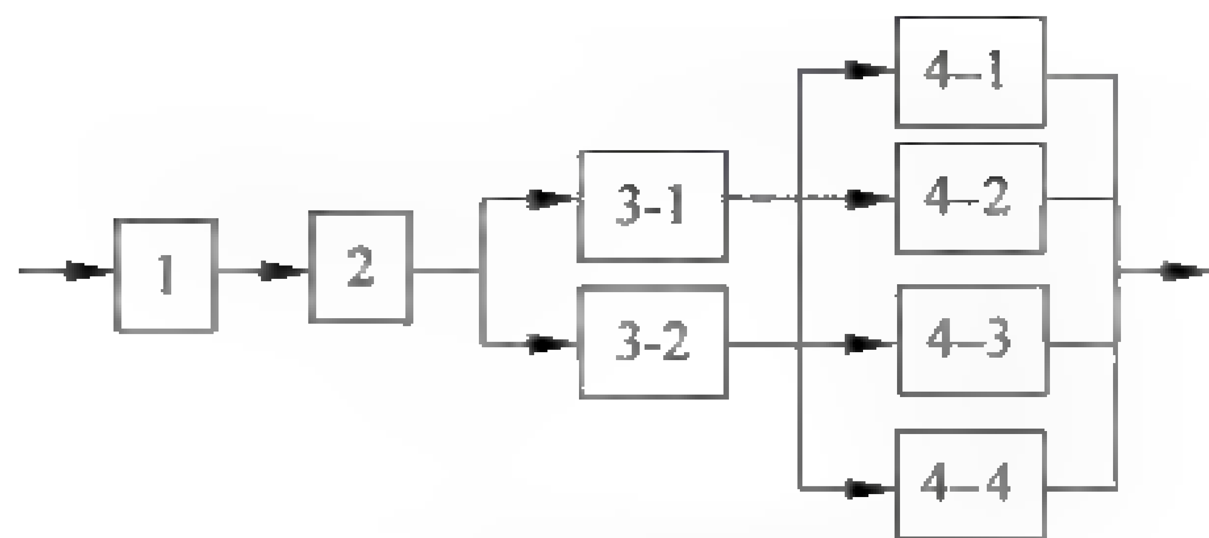


图 3.8　重复设置部件

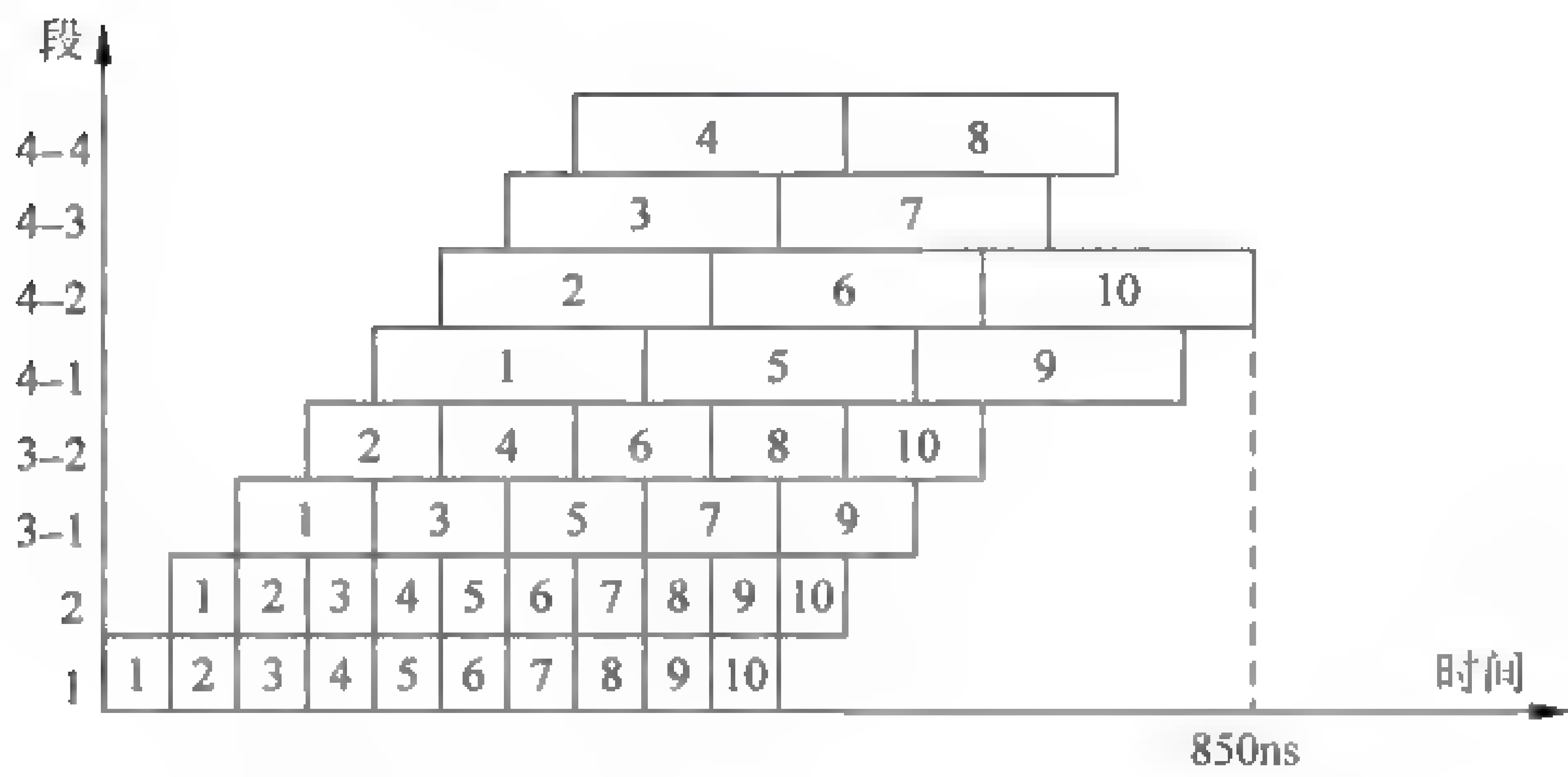


图 3.9　时空图

$$E=\frac{400\times 10}{850\times 8}=\frac{10}{17}\approx 58.82\%$$

【题 3.34】

解：(1) 会发生流水线阻塞情况，如表 3.7 所示。

表 3.7　流水线阻塞情况

第 1 个任务	S_1	S_2	S_3	S_3	S_4						
第 2 个任务		S_1	S_2	stall	S_3	S_3	S_4				
第 3 个任务			S_1	stall	S_2	stall	S_3	S_3	S_4		
第 4 个任务					S_1	stall	S_2	stall	S_3	S_3	S_4

(2) 不阻塞情况。

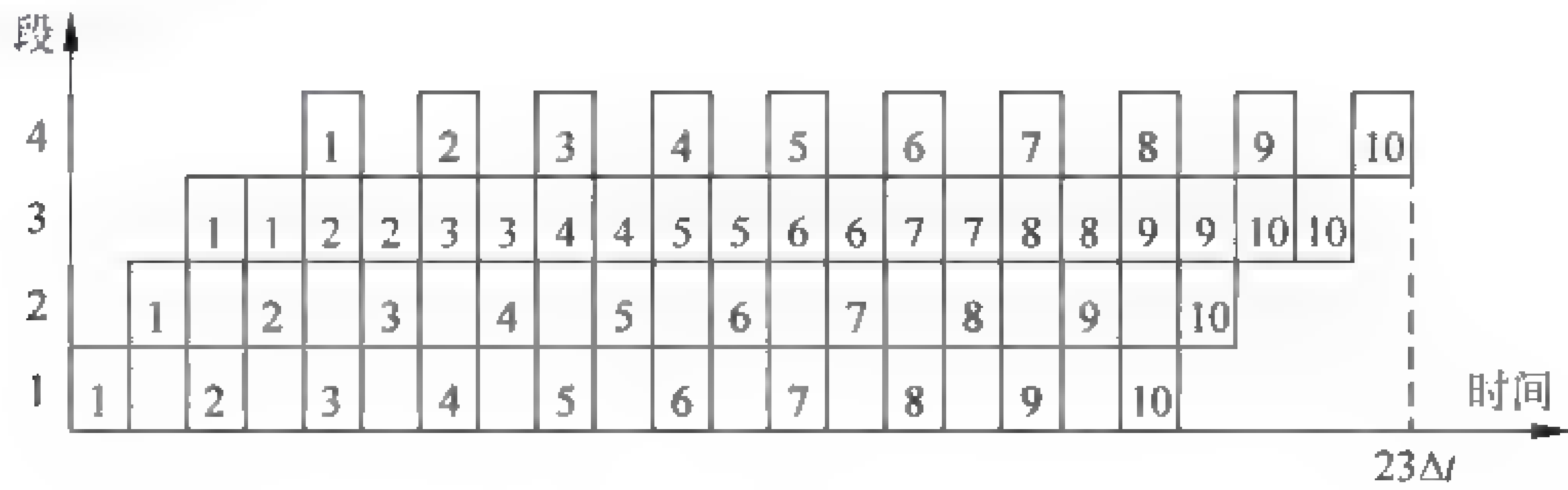


图 3.10　时空图

$$TP_{max} = \frac{1}{2\Delta t}$$

$$T_{pipeline} = 23\Delta t$$

$$TP = \frac{n}{T_{pipeline}} = \frac{10}{23\Delta t}$$

$$E = TP \cdot \frac{5\Delta t}{4} = \frac{50}{92} \approx 54.35\%$$

(3) 重复设置部件。

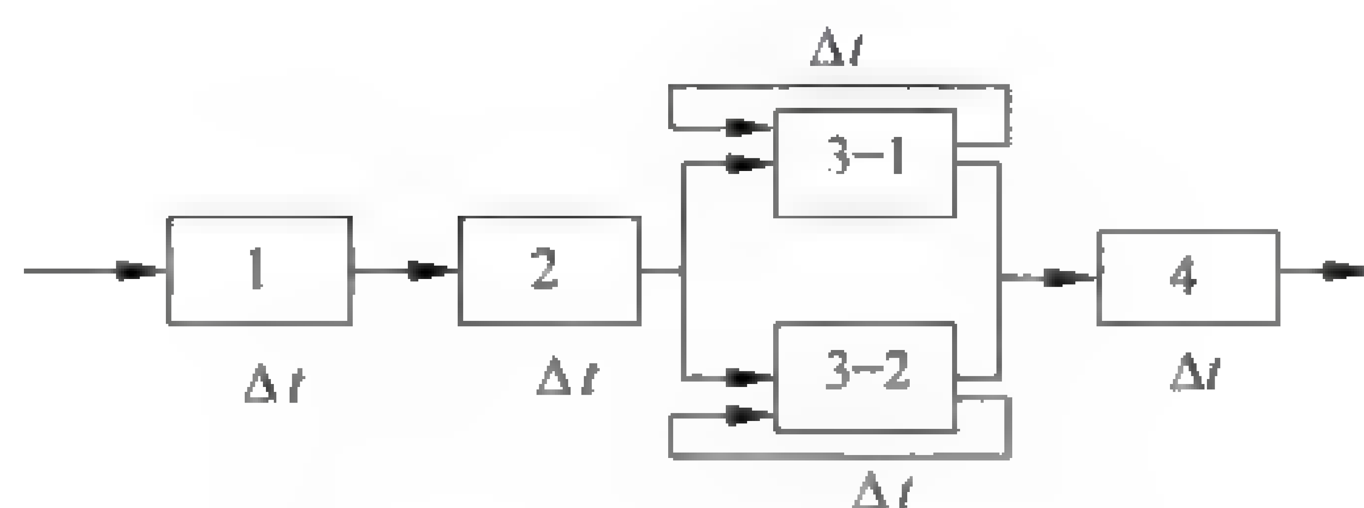


图 3.11　重复设置部件

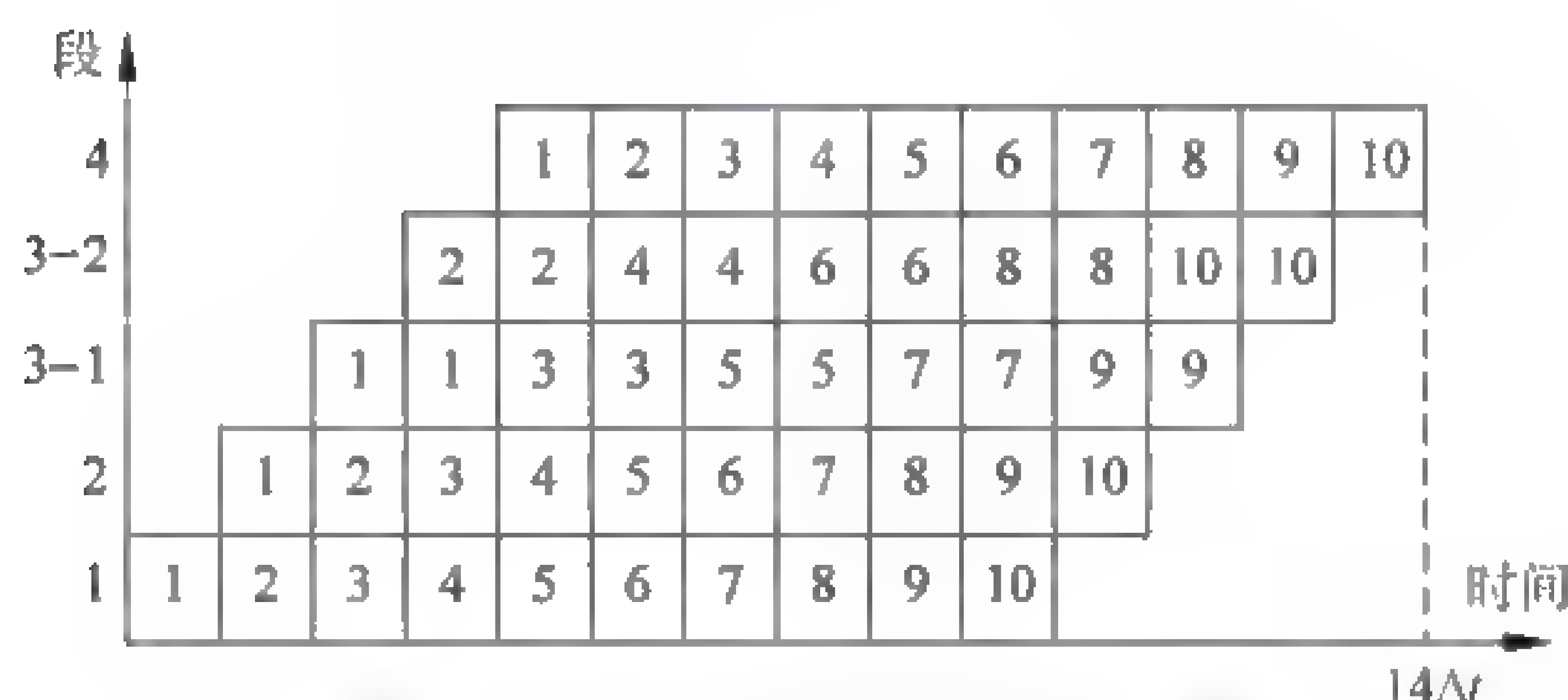


图 3.12　重复设置部件后的时空图

$$TP = \frac{n}{T_{pipeline}} = \frac{10}{14 \cdot \Delta t} = \frac{5}{7 \cdot \Delta t}$$

$$\text{吞吐率提高倍数} = \frac{\frac{5}{7\Delta t}}{\frac{10}{23\Delta t}} = 1.64$$

【题 3.35】

解：首先，应选择适合于流水线工作的算法。对于本题，应先计算 A_1+B_1、A_2+B_2、A_3+B_3 和 A_4+B_4；再计算 $(A_1+B_1)\times(A_2+B_2)$ 和 $(A_3+B_3)\times(A_4+B_4)$；然后求总的结果。

其次，画出完成该计算的时空图，如图 3.13 所示，图中阴影部分表示该段在工作。

由图 3.13 可见，它在 18 个 Δt 时间中，给出了 7 个结果。所以吞吐率为：

$$TP = \frac{7}{18\Delta t}$$

如果不用流水线，由于一次求积需 $3\Delta t$，一次求和需 $5\Delta t$，则产生上述 7 个结果共需 $(4\times5+3\times3)\Delta t=29\Delta t$。所以加速比为：

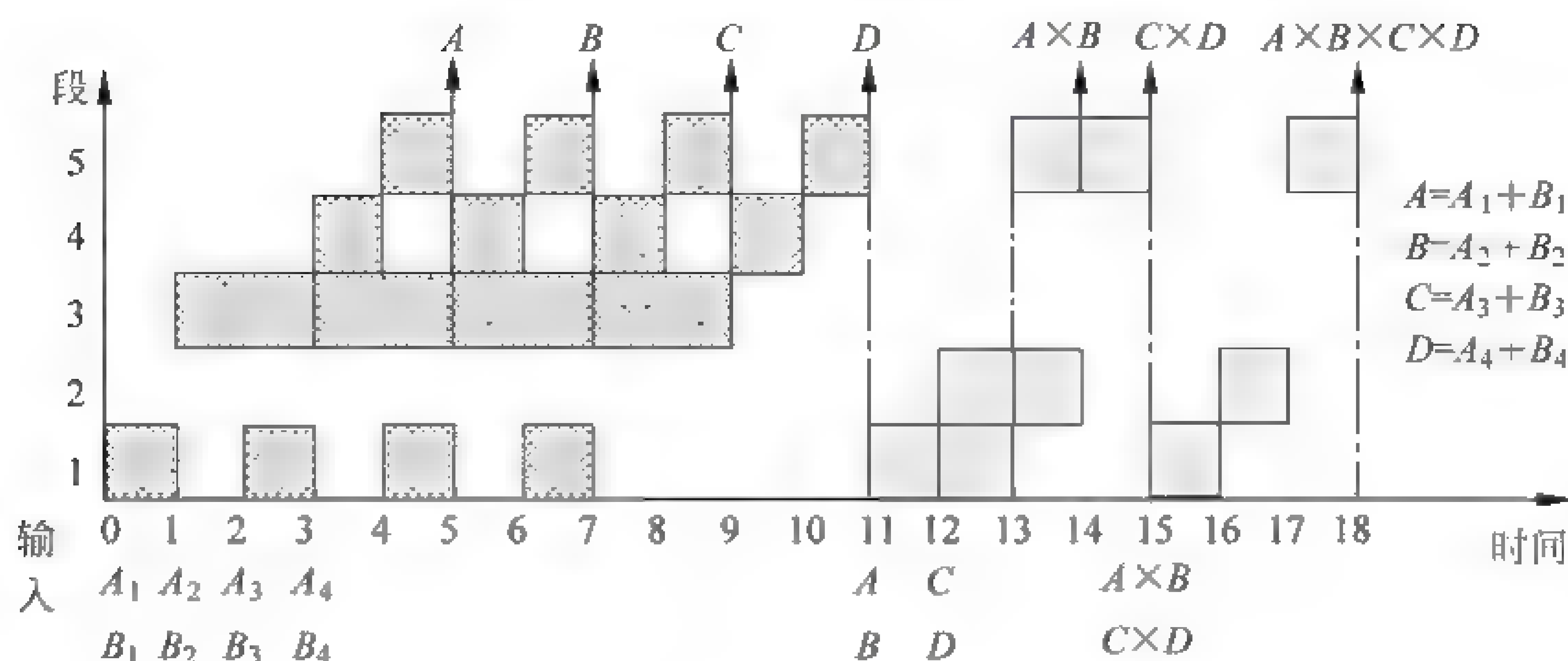


图 3.13 时空图

$$S=\frac{29\Delta t}{18\Delta t}=1.61$$

该流水线的效率可由阴影区的面积和5个段总时空区的面积的比值求得：

$$E=\frac{4\times5+3\times3}{5\times18}=0.322$$

【题 3.36】

解：首先，应选择适合于流水线工作的算法。对于本题，应先计算 $A_1\times B_1$、$A_2\times B_2$、$A_3\times B_3$ 和 $A_4\times B_4$；再计算$(A_1\times B_1)+(A_2\times B_2)$和$(A_3\times B_3)+(A_4\times B_4)$；然后求总的累加结果。

其次，画出完成该计算的时空图，如图3.14所示，图中阴影部分表示该段在工作。

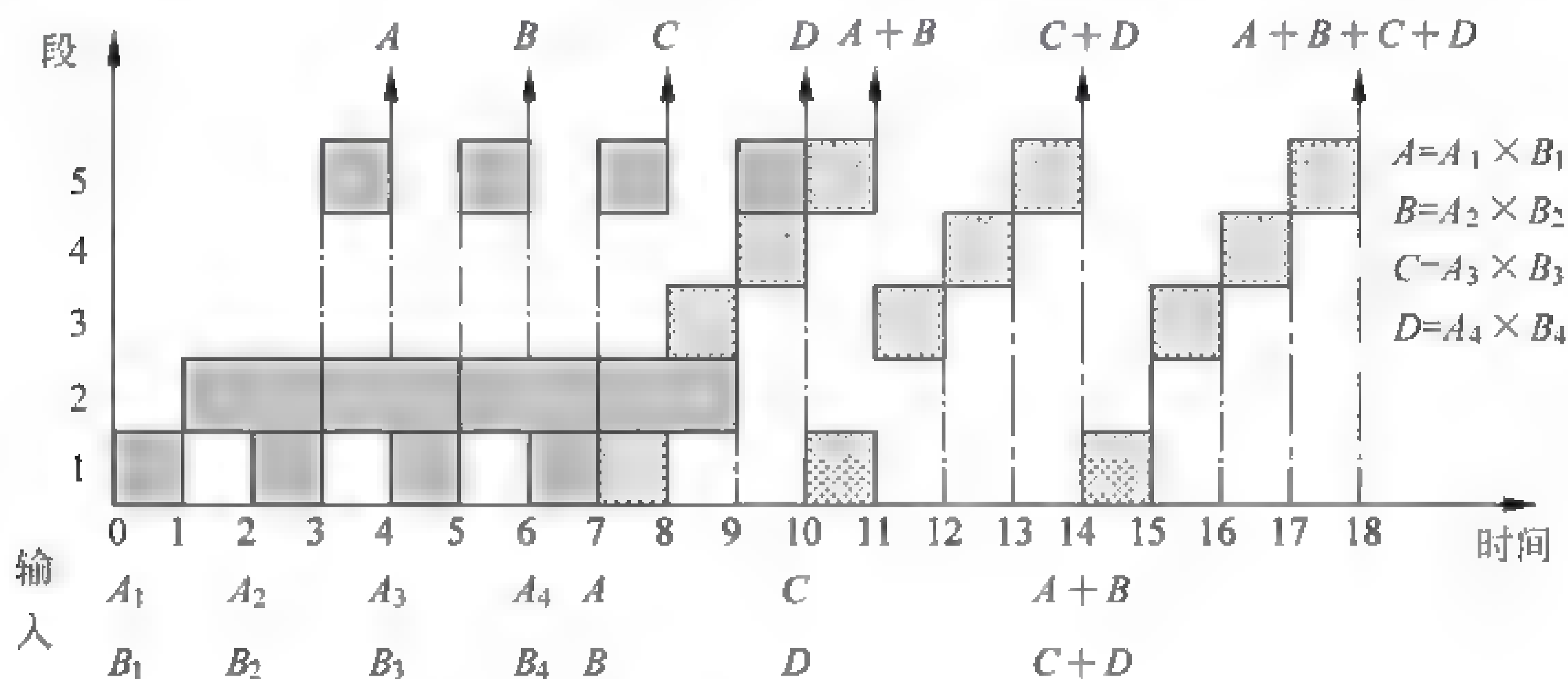


图 3.14 时空图

由图3.14可见，它在18个Δt时间中，给出了7个结果。所以吞吐率为：

$$TP=\frac{7}{18\Delta t}$$

如果不用流水线，由于一次求积需$4\Delta t$，一次求和需$4\Delta t$，则产生上述7个结果共需$(4\times4+3\times4)\Delta t=28\Delta t$。所以加速比为：

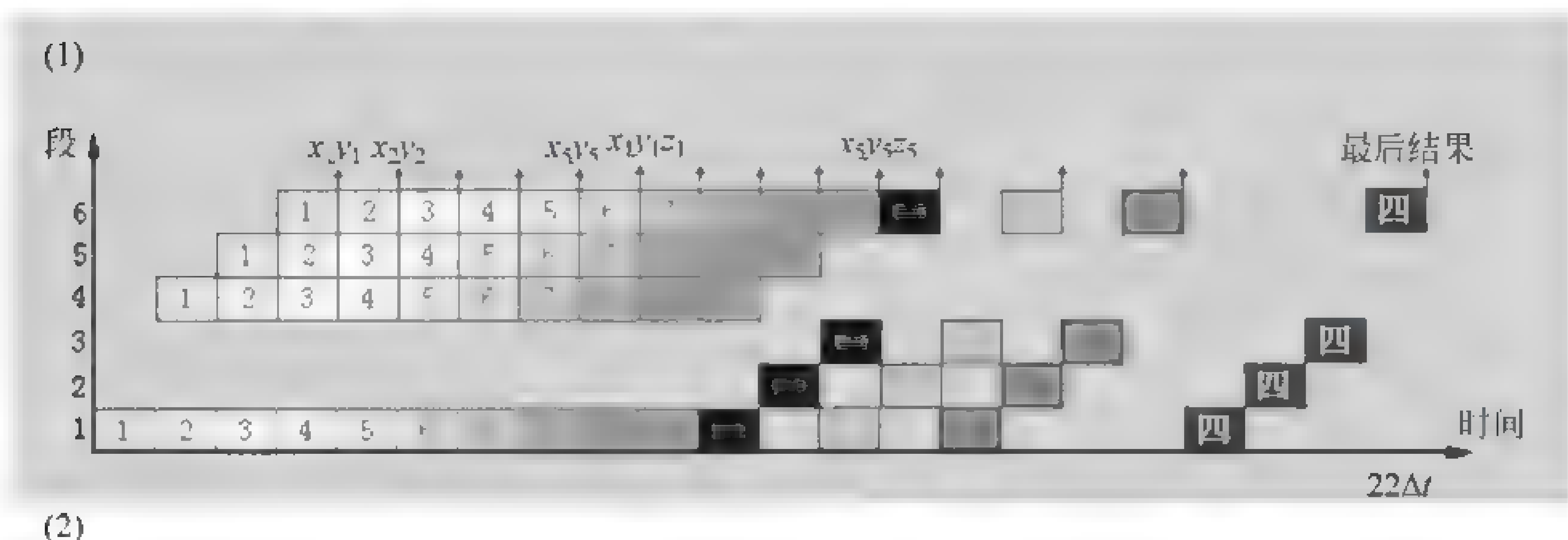
$$S=\frac{28\Delta t}{18\Delta t}\approx1.56$$

该流水线的效率可由阴影区和5个段总时空区的比值求得：

$$E=\frac{4\times 4+3\times 4}{5\times 18}\approx 0.31$$

【题 3.37】

解：机器一共要做 10 次乘法，4 次加法。时空图如图 3.15 所示。



(2)

$$TP=\frac{14}{22\Delta t}$$

$$加速比=\frac{14\times 4}{22\Delta t}=2.55$$

$$效率=\frac{14\times 4}{22\times 6}=42.42\%$$

图 3.15　时空图

【题 3.38】

解：流水线的预约表如表 3.8 所示。

表 3.8　流水线的预约表

功能段 \ 时间	1	2	3	4	5	6
S_1	√				√	
S_2		√				√
S_3			√			
S_4				√		

【题 3.39】

解：(1) 由预约表得出禁止表：$F=\{8,4,3,1\}$

为避免争用 S_1 段，禁用启动距离：8；为避免争用 S_2 段，禁用启动距离：1；为避免争用 S_3 段，禁用启动距离：3，4，1；为避免争用 S_4 段，禁用启动距离：1；为避免争用 S_5 段，禁用启动距离：1。

由禁止表得到初始冲突向量：$C_0=(10001101)$

$C_0=(10001101)$有 4 个后继状态：

$C_1=SHR^{(2)}(C_0)\vee C_0=(00100011)\vee(10001101)=(10101111)$

$C_2 = \mathrm{SHR}^{(5)}(C_0) \vee C_0 = (00000100) \vee (10001101) = (10001101) = C_0$

$C_3 = \mathrm{SHR}^{(6)}(C_0) \vee C_0 = (00000010) \vee (10001101) = (10001111)$

$C_4 = \mathrm{SHR}^{(7)}(C_0) \vee C_0 = (00000001) \vee (10001101) = (10001101) = C_0$

$C_1 = (10101111)$有两个后继状态：

$C_5 = \mathrm{SHR}^{(5)}(C_1) \vee C_0 = (10001101) = C_0$

$C_6 = \mathrm{SHR}^{(7)}(C_1) \vee C_0 = (10001101) = C_0$

$C_3 = (10101111)$有三个后继状态：

$C_7 = \mathrm{SHR}^{(5)}(C_3) \vee C_0 = (10001101) = C_0$

$C_8 = \mathrm{SHR}^{(6)}(C_3) \vee C_0 = (10001111) = C_3$

$C_9 = \mathrm{SHR}^{(7)}(C_3) \vee C_0 = (10001101) = C_0$

得出状态转移图如图 3.16 所示。

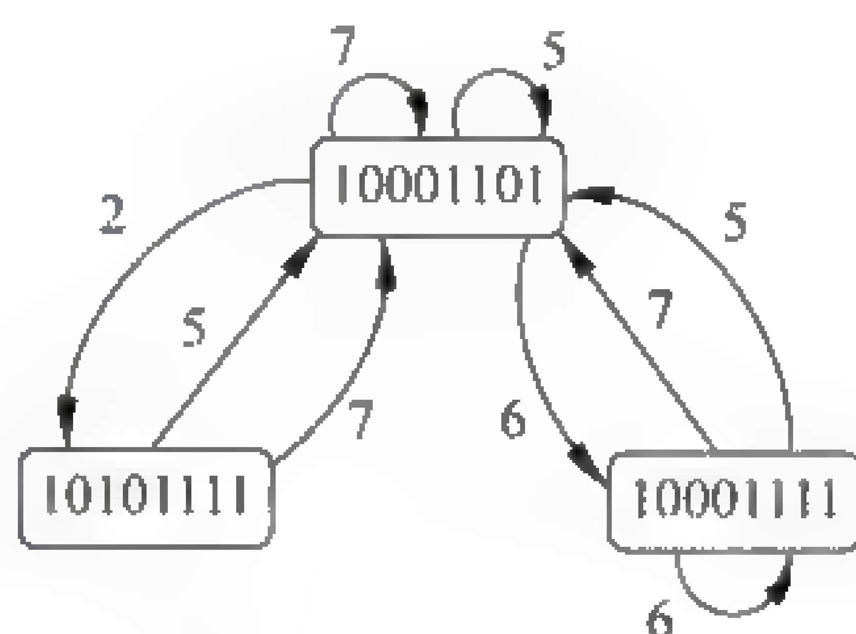


图 3.16　状态转移图

各种调度策略及平均延迟时间如表 3.9 所示。

表 3.9　调度策略及平均延迟时间

调度策略	平均延迟时间
(2,5)	$3.5\Delta t$
(2,7)	$4.5\Delta t$
(5)	$5\Delta t$
(6,5)	$5.5\Delta t$
(6)	$6\Delta t$
(6,7)	$6.5\Delta t$
(7)	$7\Delta t$

(2) 在表 3.9 中，平均延迟时间最小的调度策略是最优调度策略：(2,5)；流水线的最大吞吐率就是最优调度策略的最大吞吐率：$1/(3.5\Delta t)$。

(3) 按最优调度策略连续输入 6 个任务，流水线的实际吞吐率为：

$$\mathrm{TP} = \frac{6}{(2+5+2+5+2+9)\Delta t} = \frac{6}{25\Delta t}$$

【题 3.40】

解：(1) 由预约表得出禁止表：$F = \{6,3,1\}$

为避免争用 S_1 段，禁用启动距离：6；为避免争用 S_2 段，禁用启动距离：3；为避免争用

S_3 段，禁用启动距离：1；为避免争用 S_4 段，禁用启动距离：3；为避免争用 S_5 段，禁用启动距离：1。

由禁止表得到初始冲突向量：$C_0 = (100101)$，由初始冲突向量和后继冲突向量的计算公式 $C_j = \mathrm{SHR}^{(k)}(C_i) \vee C_0$，可得流水线任务调度的状态转移图如图 3.17 所示。

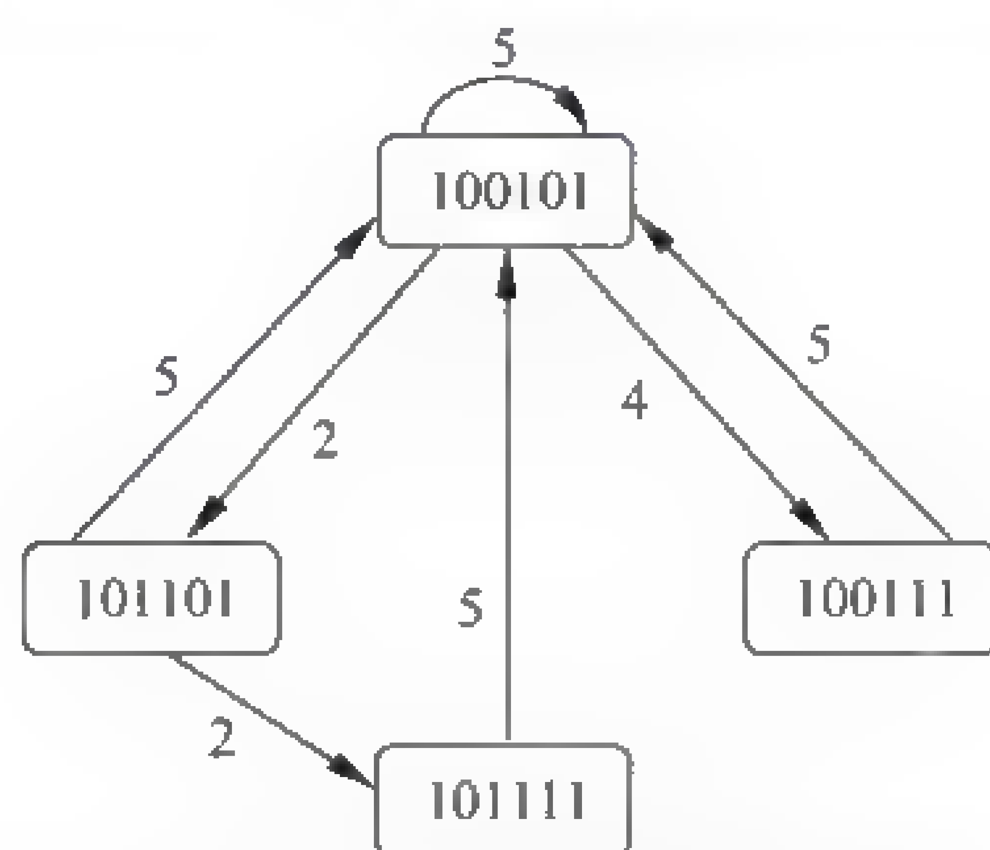


图 3.17　流水线任务调度的状态转移图

(2) 由状态转移图可得不发生段争用冲突的调度策略以及平均延迟时间，如表 3.10 所示。

表 3.10　调度策略和平均延迟时间

调度策略	平均延迟时间
(2,2,5)	$3\Delta t$
(2,5)	$3.5\Delta t$
(4)	$4\Delta t$
(4,5)	$4.5\Delta t$
(5)	$5\Delta t$

由表 3.10 可知，允许不等时间间隔调度的最优调度策略是(2,2,5)，流水线最大吞吐率为：$1/(3\Delta t)$。等时间间隔调度的最优调度策略是(4)，流水线最大吞吐率为：$1/(4\Delta t)$。

(3) 按调度策略(2,2,5)，连续输入 10 个任务的流水线实际吞吐率与加速比分别为：

$$\mathrm{TP}_1 = \frac{10}{(2+2+5+2+2+5+2+2+5+7)\Delta t} = \frac{10}{34\Delta t}$$

$$S_1 = \frac{10 \times 7\Delta t}{34\Delta t} = 2.06$$

按调度策略(4)，连续输入 10 个任务的流水线实际吞吐率与加速比分别为：

$$\mathrm{TP}_2 = \frac{10}{(4 \times 9 + 7)\Delta t} = \frac{10}{43\Delta t}$$

$$S_2 = \frac{10 \times 7\Delta t}{43\Delta t} = 1.63$$

【题 3.41】

解：

(1) 寄存器读写可以定向，无其他旁路硬件支持。排空流水线时空图如图 3.18 所示。

指令	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LW	IF	ID	EX	M	WB																	
DADDIU		IF	S	S	ID	EX	M	WB														
SW					IF	S	S	ID	EX	M	WB											
DADDIU								IF	ID	EX	M	WB										
DSUB									IF	S	S	ID	EX	M	WB							
BNEZ												IF	S	S	ID	EX	M	WB				
LW															IF	S	S	IF	ID	EX	M	WB

图 3.18　排空流水线时空图

第 i 次迭代(i=0…98)开始周期：1+(i×17)

(2) 有正常定向路径，预测分支失败，如图 3.19 所示。

指令	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LW	IF	ID	EX	M	WB										
DADDIU		IF	ID	S	EX	M	WB								
SW			IF	S	ID	EX	M	WB							
DADDIU					IF	ID	EX	M	WB						
DSUB						IF	ID	EX	M	WB					
BNEZ							IF	ID	EX	M	WB				
LW								IF	miss	miss	IF	ID	EX	M	WB

图 3.19　采用预测分支失败的时空图

第 i 次迭代(i=0…98)开始周期：1+(i×10)

总的时钟周期数：(98×10)+11=991

(3) 有正常定向路径。单周期延迟分支的时空图如图 3.20 所示。

```
LOOP:  LW      R1,0(R2)
       DADDIU  R2,R2,#4
       DADDIU  R1,R1,#1
       DSUB    R4,R3,R2
       BNEZ    R4,LOOP
       SW      R1,-4(R2)
```

第 i 次迭代(i =0…98)开始周期：1+(i×6)

总的时钟周期数：(98×6)+10=598

指令	1	2	3	4	5	6	7	8	9	10	11
LW	IF	ID	EX	M	WB						
DADDIU		IF	ID	EX	M	WB					
DADDIU			IF	ID	EX	M	WB				
DSUB				IF	ID	EX	M	WB			
BNEZ					IF	ID	EX	M	WB		
SW						IF	ID	EX	M	WB	
LW							IF	ID	EX	M	WB

图 3.20　采用单周期延迟分支的时空图

【题 3.42】

解：没有控制相关时流水线的平均 CPI=1。

存在控制相关时，由于无条件分支在第二个时钟周期结束时就被解析出来，而条件分支要到第 3 个时钟周期结束时才能被解析出来。所以：

(1) 若使用排空流水线的策略，则对于条件分支，有两个额外的停顿，对无条件分支，有一个额外的停顿：

$$CPI = 1 + 20\% \times 2 + 5\% \times 1 = 1.45$$

$$\text{加速比 } S = CPI/1 = 1.45$$

(2) 若使用预测分支成功策略，则对于不成功的条件分支，有两个额外的停顿，对无条件分支和成功的条件分支，有一个额外的停顿：

$$CPI = 1 + 20\% \times (60\% \times 1 + 40\% \times 2) + 5\% \times 1 = 1.33$$

$$\text{加速比 } S = CPI/1 = 1.33$$

(3) 若使用预测分支失败策略，则对于成功的条件分支，有两个额外的停顿；对无条件分支，有一个额外的停顿；对不成功的条件分支，其目标地址已经由 PC 值给出，不必等待，所以无延迟：

$$CPI = 1 + 20\% \times (60\% \times 2 + 40\% \times 0) + 5\% \times 1 = 1.29$$

$$\text{加速比 } S = CPI/1 = 1.29$$

第 4 章　向量处理机

4.1　基本要求与难点

4.1.1　基本要求

(1) 掌握有关向量处理机的基本概念。
(2) 掌握向量处理机的 3 种方式及其特点。
(3) 理解向量处理机的两种结构。
(4) 掌握向量处理机上一个向量指令序列是否能并行执行或链接执行的判断方法，并能计算指令序列的总执行时间。
(5) 掌握向量处理机的性能指标的定义及其计算方法。
(6) 了解 Cray Y-MP、C-90 和 NECSX-X44 向量处理机的结构。

4.1.2　难点

(1) 向量处理机上一个向量指令序列是否能并行执行或链接执行的判断。该指令序列的执行时间的计算。
(2) 向量处理机的性能指标的定义及其计算方法。

4.2　知识要点

向量处理机：设置了向量数据表示和向量指令的流水线处理机。
标量处理机：不具有向量数据表示和向量指令的处理机。

4.2.1　向量的处理方式

1. 横向处理方式

在横向处理方式中，向量计算是按行的方式从左到右横向地进行。
这种处理方式只适合于一般的处理机，不适合于向量处理机的并行处理。

2. 纵向处理方式

在这种方式中，向量计算是按列的方式从上到下纵向地进行。也就是说，是将整个向量按相同的运算处理完之后，再去进行别的运算。这种处理方式适用于向量处理机，对向量长度 N 的大小没有限制。

3. 纵横处理方式

纵横处理方式又称为分组处理方式，是上述两种方法的一种结合。它是把向量分成若干组，组内按纵向方式处理，依次处理各组。这种处理方式对向量长度 N 的大小也不限制，但它是以每 n 个元素为一组进行分组处理的。n 的值是固定的。

这种处理方式也适用于向量处理机。

4.2.2 向量处理机的结构

向量处理机的结构主要是由其所采用的向量处理方式决定的。有两种典型的结构：存储器-存储器型结构，寄存器-寄存器型结构。纵向处理方式宜采用前者，而分组处理方式则宜采用后者。

1. "存储器-存储器"结构

在纵向处理方式中，向量长度 N 的大小不受限制，无论 N 有多大，相同的运算都用一条向量指令完成。在这种向量处理机中，流水线运算部件的输入和输出端都直接(或经过缓冲器)与存储器相连，从而构成"存储器-存储器"型操作的运算流水线。

这种结构对存储器的带宽以及存储器与处理部件的通信带宽提出了非常高的要求。一般是通过采用多体交叉并行存储器和缓冲器技术来解决。

2. "寄存器-寄存器"结构

在向量的分组处理方式中，向量被分为每 n 个数据一组。以组为单位进行数据的存取和运算。组的长度 n 是固定不变的，所以可以设置能快速访问的向量寄存器，用于存放源向量、目的向量及中间结果。让运算部件的输入、输出端都与向量寄存器相连，就构成了"寄存器-寄存器"型操作的运算流水线。

Cray 1 是美国 CRAY 公司于 1976 年推出的巨型计算机，它的时钟周期是 12.5ns，其浮点运算速度达到了 1 亿次/秒以上。它是典型的采用"寄存器-寄存器"结构的向量机。它由以下几个部分构成。

(1) 功能部件。共有 12 条可并行工作的单功能流水线，可分别流水地进行地址、向量、标量的各种运算。

(2) 向量寄存组 V。由 512 个 64 位的寄存器组成，分成 8 组，编号为 $V_0 \sim V_7$。每一个组称为一个向量寄存器。

(3) 标量寄存器 S 和快速暂存器 T。标量寄存器有 8 个：$S_0 \sim S_7$，都是 64 位。

(4) 向量屏蔽寄存器 VM。64 位，每一位对应于向量寄存器的一个单元。其作用是用

于向量的归并、压缩、还原和测试操作等，也可用于实现对向量某些元素的单独运算。

为了能充分发挥向量寄存器和可并行工作的6个流水线功能部件的作用，加快对向量的处理，Cray-1中的每个向量寄存器V_i都有连到6个向量功能部件的单独总线，而每个向量功能部件也都有把运算结果送回向量寄存器组的总线。这样，只要不出现V_i冲突和功能部件冲突，各V_i之间和各功能部件之间都能并行工作，大大加快了向量指令的处理，这是Cray-1向量处理的一个显著特点。

Cray-1的指令系统包括标量类和向量类指令共128条，其中3种向量指令为：①"向量-向量"之间的运算，从两个向量寄存器V_j、V_k取得源向量，进行运算操作后结果送入向量寄存器V_i；②"向量-标量"之间的运算，它与第①种指令的区别在于它的一个操作数取自标量寄存器S_j；第③种指令用于实现主存与向量寄存器组之间的成组数据传送。

4.2.3　提高向量处理机性能的常用技术

为了提高向量处理机的性能，可采用多种方法。

(1) 设置多个功能部件，使它们并行工作；

(2) 采用链接技术，加快一串向量指令的执行；

(3) 采用循环开采技术，加快循环的处理；

(4) 采用多处理机系统，进一步提高性能。

1. 设置多个功能部件

在向量处理机中，为了提高性能，通常都设置多个独立的功能部件。这些部件能按流水方式并行工作，从而形成了多条并行工作的运算操作流水线。

2. 链接技术

当流出向量指令时，需要占用相关的功能流水线和向量寄存器。只有等该指令执行完毕，才可以释放功能流水线和向量寄存器。占用多少时间取决于向量的长度和流水线延迟。

(1) 当两条向量指令之间既没有V_i冲突，也没有功能部件冲突时，可以并行执行。

V_i冲突是指多条向量指令的源向量或结果向量要使用相同的V_i，功能部件冲突是指多条向量指令要使用同一个功能部件。

(2) 当两条向量指令发生功能部件冲突时，后一条指令要等前一条执行完成、释放功能部件后才能流出。

(3) 当两条向量指令发生V_i冲突时，要等前一条执行完成、释放寄存器后，后一条指令才能流出。

(4) 当前一条指令的结果寄存器是后一条指令的源寄存器，且不存在其他冲突时，就可以用链接技术来提高性能。

向量流水线链接是指：具有先写后读相关的两条指令，在不出现功能部件冲突和V_i冲突的情况下，可以把功能部件链接起来进行流水处理，以达到加快执行的目的。具体来说，就是当前一个向量功能部件产生第一个结果并送到结果向量寄存器的入口时，将该结果立即送往下一个功能部件的入口，开始后续的向量处理操作。此后依次得到的中间结果都按

此处理。这样，前面功能部件的结果元素一产生，就可以立即被后面功能部件所使用，而不用等结果向量全部产生后再来使用。

链接特性是 Cray-1 向量处理的一个显著特点。

由于同步的要求，链接时，Cray-1 中把向量数据元素送往向量功能部件以及把结果存入向量寄存器都需要一拍时间，从存储器中把数据送入访存功能部件也需要一拍时间。

进行向量链接时，除了要求无 V_i 冲突和无功能部件使用冲突外，还有一些别的要求。

(1) 只有在前一条指令的第一个结果元素送入结果向量寄存器的那一个时钟周期才可以进行链接。如果错过这个时刻，就无法进行链接了，这时只好按串行方式执行这两条指令。

(2) 当一条向量指令的两个源操作数分别来自前面紧邻的两条指令的结果时，要求这两条指令产生运算结果的时间必须相等，即要求有关功能部件的通过时间相等。

(3) 要进行链接执行的向量指令的向量长度必须相等，否则无法进行链接。

3. 分段开采技术

当向量的长度大于向量寄存器的长度时，必须把长向量分成长度固定的段，然后循环分段处理，每一次循环只处理一个向量段。这种技术称为分段开采技术。将长向量分段成循环处理是由系统自动完成的，对程序员是透明的。

4. 采用多处理机系统

为了进一步提高系统的向量处理性能，许多新型向量处理机系统采用了多处理机系统结构。例如，在 Cray-1 的基础上，CRAY 公司在 20 世纪 80 年代推出了 Cray-2、Cray X-MP，在 20 世纪 90 年代推出了 Cray Y-MP、C90，这些机器基本上保持了 Cray-1 的基本结构，但都已经发展成为多处理机系统。Cray-2 包含 4 个向量处理机，浮点运算速度最高可达 1800MFLOPS。Cray Y-MP、C90 最多可包含 16 个向量处理机。

4.2.4　向量处理机的性能评价

衡量向量处理机性能的主要参数有：向量指令的处理时间 T_{vp}，向量长度为无穷大时的向量处理机的最大性能 R_{∞}，半性能向量长度 $n_{1/2}$，向量长度临界值 n_v。

1. 向量指令的处理时间 T_{vp}

1) 一条向量指令的执行时间 T_{vp}

$$T_{vp} = (T_{start} + n)T_c \tag{4.1}$$

其中，T_{start} 为从一条向量指令开始执行到还差一个时钟周期就产生第一个结果所需的时钟周期数。可称之为该向量指令的启动时间。此后，便是每个时钟周期流出一个结果，共有 n 个结果。

2) 一组向量指令的总执行时间

一组向量指令的执行时间主要取决于 3 个因素：向量的长度、向量操作之间是否存在流水功能部件的使用冲突以及数据的相关性。我们把能在同一个时钟周期内一起开始执行

的几条向量指令称为一个编队。同一个编队中的向量指令之间一定不存在流水功能部件的冲突或数据的相关性。如果存在这种冲突或相关,那么就必须将它们编入不同的编队。

编队后,这个向量指令序列的总的执行时间为:

$$T_{all} = (T_{start} + mn)T_c \tag{4.2}$$

其中,$T_{start} = \sum_{i=1}^{m} T_{start}^{(i)}$ 是该组指令总的启动时间(时钟周期个数),$T_{start}^{(i)}$ 表示第 i 编队中各指令的启动时间的最大值,m 为编队的个数,n 为向量的长度。

如果表示成时钟周期个数,则

$$T_{all} = T_{start} + mn \tag{4.3}$$

3) 分段开采时一组向量指令的总执行时间

$$T_{all} = \left\lceil \frac{n}{\text{MVL}} \right\rceil \times (T_{start} + T_{loop}) + mn \tag{4.4}$$

T_{loop}为循环所引入的额外时间(时钟周期数)。

2. 最大性能 R_∞ 和半性能向量长度 $n_{1/2}$

最大性能 R_∞ 表示当向量长度为无穷大时,向量处理机的最大性能,也称为峰值性能。单位是 MFLOPS。

$$R_\infty = \lim_{n \to \infty} \frac{\text{向量指令序列中浮点运算次数} \times \text{时钟频率}}{\text{向量指令序列执行所需的时钟周期数}} \tag{4.5}$$

半性能向量长度 $n_{1/2}$是指向量处理机的性能为其最大性能 R_∞ 的一半时所需的向量长度。它是评价向量流水线的建立时间对性能影响的重要参数。若向量长度 $n=n_{1/2}$,则表明整个向量流水处理时间中只有一半是在做有效操作,而另一半是浪费掉了。

通常都希望向量处理机有较小的 $n_{1/2}$。实际测试表明,Cray 1 的 $n_{1/2}$ 为 10～20,Cyber 205 的 $n_{1/2}$ 为 100。这表明 Cray 1 的流水线建立时间比 Cyber 205 的要小很多。

3. 向量长度临界值 n_v

向量长度临界值 n_v 是指向量流水方式的处理速度优于标量串行方式的处理速度时所需的向量长度的最小值。该参数既衡量建立时间,也衡量标量、向量处理速度比对向量处理机性能的影响。

习　　题

1. 概念题

【题 4.1】 解释下列名词

横向处理方式	纵向处理方式	纵横处理方式	向量流水线链接
V_i冲突	功能部件冲突	分段开采技术	半性能向量长度
向量长度临界值			

2. 填空题

【题 4.2】 向量流水处理机采用________结构或________结构。

【题 4.3】 Cray-1 向量处理的一个显著特点是：只要不出现________冲突和________冲突，各 V_i 之间和各功能部件之间都能并行工作。

【题 4.4】 衡量向量处理机性能的主要参数有________、________、________和________。

【题 4.5】 把能在同一个时钟周期内一起开始执行的几条向量指令称为一个________。

【题 4.6】 在向量流水处理机上，向量指令序列中的一个编队内的指令可以________执行，编队执行时间为编队内所有的向量指令执行时间的________。

3. 选择题

【题 4.7】 Cray-1 的流水线是(　　)。

A. 多条单功能流水线　　B. 一条单功能流水线

C. 多条多功能流水线　　D. 一条多功能流水线

【题 4.8】 Cray-1 向量处理机要实现指令间的链接，必须满足下列条件中的(　　)。

A. 源向量相同，功能部件不冲突，有指令相关

B. 源向量不同，功能部件相同，无指令相关

C. 源向量、功能部件都不相同，指令有写后读冲突

D. 源向量、功能部件都不相同，指令有读后写冲突

【题 4.9】 Cray-1 向量处理机启动存储器、流水部件及寄存器打入各需一拍，现有向量指令串：

$V_3 \leftarrow$ 存储器　　(从存储器中取数：6 拍)

$V_4 \leftarrow V_0 + V_1$　　(向量加：6 拍)

$V_5 \leftarrow V_3 \times V_4$　　(向量乘：7 拍)

向量长度均为 N，则指令串最短的执行时间是(　　)。

A. $16+N$ 拍　　B. $17+N$ 拍　　C. $18+N$ 拍　　D. $19+N$ 拍

【题 4.10】 Cray-1 的两条向量指令：

$V_1 \leftarrow V_2 + V_3$

$V_4 \leftarrow V_1 \times V_5$

属于(　　)。

A. 没有功能部件冲突和源向量冲突，可以并行

B. 没有功能部件冲突和源向量冲突，可以链接

C. 没有源向量冲突，可以交换执行顺序

D. 有向量冲突，只能串行

4. 问答题

【题 4.11】 简述 3 种向量处理方式，它们对向量处理机的结构要求有何不同？

【题 4.12】 可采用哪些方法来提高向量处理机的性能?

【题 4.13】 采用链接技术时,向量指令能够链接执行必须满足哪些条件?

5. 应用题

【题 4.14】 在 Cray-1 机器上,按照链接方式执行下述 4 条向量指令(括号中给出了相应功能部件的执行时间),如果向量寄存器和功能部件之间的数据传送需要 1 拍,试求此链接流水线的通过时间是多少拍?如果向量长度为 64,则需多少拍才能得到全部结果?

$V_0 \leftarrow$存储器 (从存储器中取数:7 拍)

$V_2 \leftarrow V_0 + V_1$ (向量加:3 拍)

$V_3 \leftarrow V_2 < A_3$ (按(A_3)左移:4 拍)

$V_5 \leftarrow V_3 \wedge V_4$ (向量逻辑乘:2 拍)

【题 4.15】 在 Cray-1 上,按照链接方式执行下述 5 条向量指令(括号中给出了相应功能部件的时间),如果向量寄存器和功能部件之间数据传输需要 1 拍,试求此链接流水线的通过时间是多少拍?如果向量长度为 64,则需要多少拍才能得到全部结果。

$V_0 \leftarrow$存储器 (从存储器中取数:7 拍)

$V_2 \leftarrow V_0 + V_1$ (向量加:3 拍)

$V_3 \leftarrow V_2 < A_3$ (按(A_3)左移:5 拍)

$V_5 \leftarrow V_3 \wedge V_4$ (向量逻辑乘:2 拍)

存储器$\leftarrow V_5$ (向存储器中存数:7 拍)

【题 4.16】 某向量处理机有 16 个向量寄存器,其中,$V_0 \sim V_5$ 中分别放有向量 A、B、C、D、E、F,向量长度均为 8,向量各元素均为浮点数;处理部件采用两条单功能流水线,加法功能部件时间为 2 拍,乘法功能部件时间为 3 拍。采用类似于 Cray 1 的链接技术,先计算 $(A+B)\times C$,在流水线不停流的情况下,接着计算 $(D+E)\times F$。

(1) 求此链接流水线的通过时间。(设寄存器入、出各需 1 拍)

(2) 假如每拍时间为 50ns,完成这些计算并把结果存进相应寄存器,此处理部件的实际吞吐率为多少 MFLOPS?

【题 4.17】 在一台向量处理机上实现 $A=B\times S$ 计算,其中,A 和 B 是长度为 $N=200$ 的向量,S 是一个标量。向量寄存器长度 MVL=64,各功能部件的启动时间为:取数和存数部件为 12 个时钟周期、乘法部件为 7 个时钟周期,执行标量代码的开销 $T_{loop}=15$ 个时钟周期,对一个向量元素执行一次操作的时间 $T_e=1$ 个时钟周期。求计算 A 的总执行时间。

【题 4.18】 向量处理机 Cray Y MP/8 的机器周期时间为 6ns,一个周期可以完成一次加和一次乘法运算。另外,8 台处理机在最好的情况下可以同时运算而互不干扰。计算 Cray Y-MP/8 的峰值性能。

【题 4.19】 A、B 两个向量存放于存储器,其向量长度为 64。设流水加法器有 4 级,流水线时钟周期为 10ns,读出 A、B 向量第一对元素到流水线始端所需的时钟周期数为 2,求执行向量加法指令 ADDV 所需的时间。

【题 4.20】 假设每种向量功能部件只有一个,而且不考虑向量链接,那么下面的一组向量指令能分成几个编队?

```
LV         V1,Rx            //取向量 x
MULTSV     V2,R0,V1         //向量 x 和标量(R0)相乘
LV         V3,Ry            //取向量 y
ADDV       V4,V2,V3         //相加,结果保存到 V4 中
SV         V4,Ry            //存结果
```

【题 4.21】 在某向量处理机上执行 DAXPY 的向量指令序列，即完成 $Y=a\times X+Y$。其中，X 和 Y 是向量，最初保存在主存中，a 是一个标量，已存放在寄存器 F0 中。它们的向量指令如下：

```
LV         V1,Rx            //取向量 X
MULTFV     V2,F0,V1         //向量 X 和标量(F0)相乘
LV         V3,Ry            //取向量 Y
ADDV       V4,V2,V3         //完成 Y = a × X + Y
SV         V4,Ry            //存结果
```

假设向量寄存器的长度 MVL=64，$T_{loop}=15$，各功能部件的启动时间为：

(1) 取数和存数部件为 12 个时钟周期；

(2) 乘法部件为 7 个时钟周期；

(3) 加法部件为 6 个时钟周期。

分别对于不采用向量链接技术和采用链接技术的两种情况，求完成上述向量操作的总执行时间。

题　　解

1. 概念题

【题 4.1】 解释下列名词

横向处理方式 —— 若向量长度为 N，则横向处理方式相当于执行 N 次循环。若使用流水线，在每次循环中可能出现数据相关和功能转换，不适合对向量进行流水处理。

纵向处理方式 —— 将整个向量按相同的运算处理完毕之后，再去执行其他运算。适合对向量进行流水处理。

纵横处理方式 —— 把长度为 N 的向量分为若干组，每组长度为 n，组内按纵向方式处理，依次处理各组，组数为$\lceil N/n \rceil$，适合流水处理。

向量流水线链接 —— 具有先写后读相关的两条指令，在不出现功能部件冲突和 V_i 冲突的情况下，可以把功能部件链接起来进行流水处理，以达到加快执行的目的。

V_i 冲突——指多条向量指令的源向量或结果向量要使用相同的 V_i。

功能部件冲突——指多条向量指令要使用同一个功能部件。

分段开采技术 —— 当向量的长度大于向量寄存器的长度时，必须把长向量分成长度固定的段，然后循环分段处理，每一次循环只处理一个向量段。

半性能向量长度 —— 指向量处理机的性能为其最大性能 R_∞ 的一半时所需的向量长度。

向量长度临界值——指向量流水方式的处理速度优于标量串行方式的处理速度时所需的向量长度的最小值。

2. 填空题

【题4.2】 答：存储器-存储器型、寄存器-寄存器型

【题4.3】 答：向量寄存器V_i、功能部件

【题4.4】 答：向量指令的处理时间、向量长度为无穷大时的向量处理机的最大性能、半性能向量长度、向量长度临界值

【题4.5】 答：编队

【题4.6】 答：同时、最大值

3. 选择题

【题4.7】 答：A

【题4.8】 答：C

【题4.9】 答：A

【题4.10】 答：B

4. 问答题

【题4.11】 答：①横向处理方式：若向量长度为N，则横向处理方式相当于执行N次循环。若使用流水线，在每次循环中可能出现数据相关和功能转换，不适合对向量进行流水处理。②纵向处理方式：将整个向量按相同的运算处理完毕之后，再去执行其他运算。适合对向量进行流水处理，向量运算指令的源/目向量都放在存储器内，使得流水线运算部件的输入、输出端直接与存储器相连，构成$M-M$型的运算流水线。③纵横处理方式：把长度为N的向量分为若干组，每组长度为n，组内按纵向方式处理，依次处理各组，组数为$\lceil N/n \rceil$，适合流水处理。可设长度为n的向量寄存器，使每组向量运算的源/目向量都在向量寄存器中，流水线的运算部件输入、输出端与向量寄存器相连，构成R R型运算流水线。

【题4.12】 答：可采用多种方法：

(1) 设置多个功能部件，使它们并行工作；

(2) 采用链接技术，加快一串向量指令的执行；

(3) 采用循环开采技术，加快循环的处理；

(4) 采用多处理机系统，进一步提高性能。

【题4.13】 答：

(1) 向量指令之间要求无V_i冲突和无功能部件使用冲突。

(2) 只有在前一条指令的第一个结果元素送入结果向量寄存器的那一个时钟周期才可以进行链接。如果错过这个时刻，就无法进行链接了。

(3) 当一条向量指令的两个源操作数分别来自前面紧邻的两条指令的结果时，要求这两条指令产生运算结果的时间必须相等，即要求有关功能部件的通过时间相等。

链接执行的向量指令的向量长度必须相等，否则无法进行链接。

5. 应用题

【题 4.14】

解：通过时间就是每条向量指令的第一个操作数执行完毕需要的时间，也就是各功能流水线由空到满的时间，具体过程如图 4.1 所示。要得到全部结果，在流水线充满之后，向量中后继操作数继续以流水方式执行，直到整组向量执行完毕。

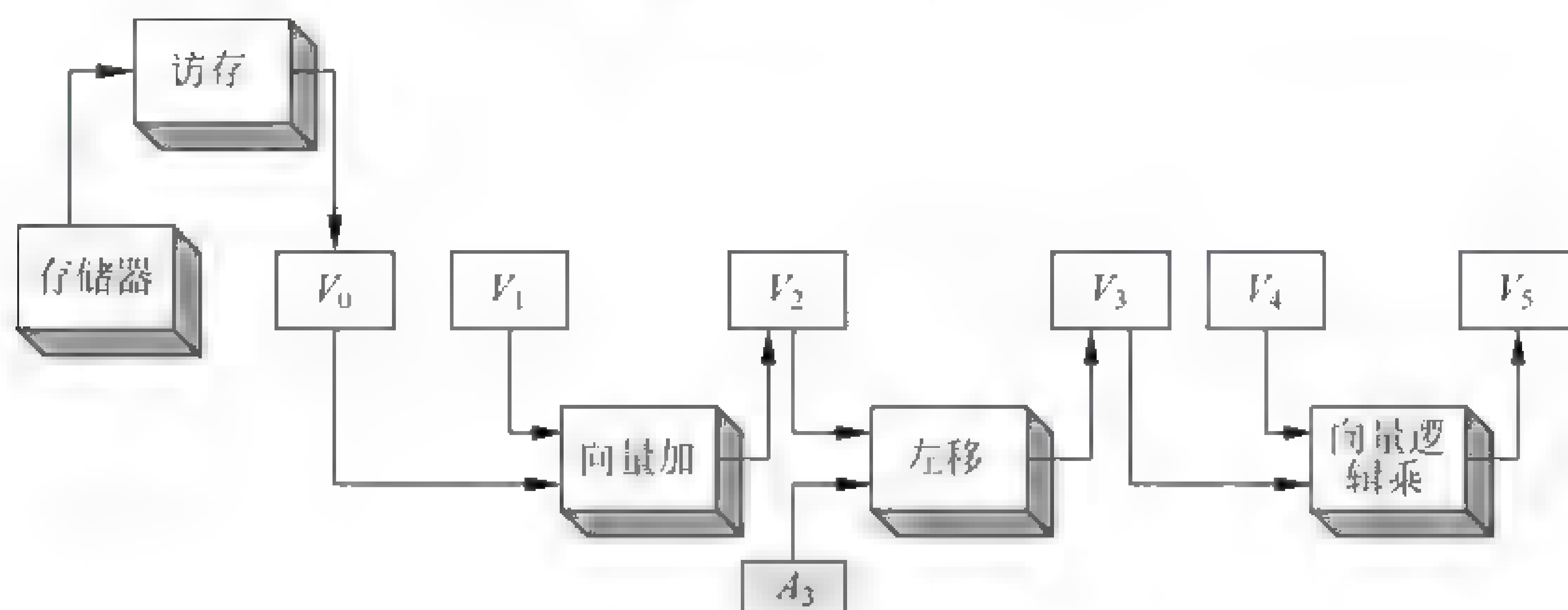


图 4.1　具体过程

$$T_{通过} = (7+1)+(1+3+1)+(1+4+1)+(1+2+1) = 23(拍)$$

$$T_{总共} = T_{通过} + (64-1) = 23+63 = 86(拍)$$

【题 4.15】

解：通过时间就是每条向量指令的第一个操作数执行完毕需要的时间，也就是各功能流水线由空到满的时间。要得到全部结果，在流水线充满之后，向量中后继操作数继续以流水方式执行，直到整组向量执行完毕。

通过时间：

$$T_1 = (7+1)+(1+3+1)+(1+5+1)+(1+2+1)+(1+7+1) = 33$$

总时间 $T_2 = T_1 + (64-1) = 96$

【题 4.16】

解：(1) 我们在这里假设 $A+B$ 的中间结果放在 V_6 中，$(A+B)\times C$ 的最后结果放在 V_7 中，$D+E$ 的中间结果放在 V_8 中，$(D+E)\times F$ 的最后结果放在 V_9 中。具体实现参考如图 4.2 所示。

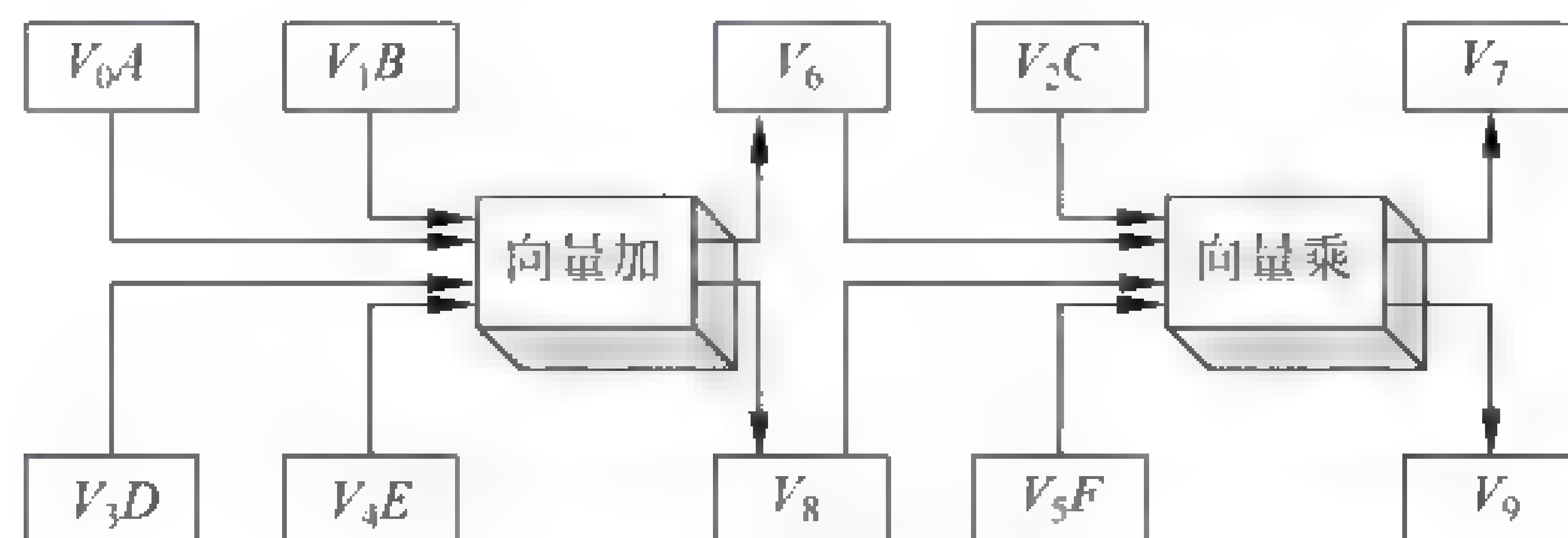


图 4.2　链接流水线

通过时间应该为前者($(A+B)\times C$)通过的时间：

$$T_{通过} = (1+2+1)+(1+3+1) = 9(拍)$$

(2) 在做完$(A+B)\times C$之后，做$(C+D)\times E$就不需要通过时间了。

$$V_6 \leftarrow A+B$$
$$V_7 \leftarrow V_6 \times C$$
$$V_8 \leftarrow D+E$$
$$V_9 \leftarrow V_8 \times F$$
$$T = T_{通过} + (8-1) + 8 = 24(拍) = 1200(\mathrm{ns})$$
$$\mathrm{TP} = \frac{32}{T} = 26.67\mathrm{MFLOPS}$$

【题 4.17】

解：假设向量A和B存放在向量寄存器Ra和Rb中，标量S存放在标量寄存器R0中，由下面3条指令完成计算：

```
LV        V1,Rb
MULTSV    V2,R0,V1
SV        V2,Ra
```

这3条指令之间存在相关，需分为3个编队$m=3$。向量需要分为$\lceil 200/64 \rceil=4$组进行计算。

由题目得：$T_{start}=12+12+7=31$；$T_{loop}=15$

$T=4\times(T_{loop}+T_{start})+3\times200\times1=4\times(15+31)+3\times200\times1=784$个时钟周期

【题 4.18】

解：Cray Y-MP/8的峰值性能为：

$$R_\infty = \frac{(1+1)\times 8}{T\times 10^6} = \frac{16}{6\times 10^{-9}\times 10^6} \approx 2667\mathrm{MFLOPS}$$

【题 4.19】

解：由题意知，$n=64$，$l=4$，$s=2$，$T_c=10\mathrm{ns}$

$$T_{vp} = (s+l+n-1)T_c = (2+4+64-1)10\mathrm{ns} = 690\mathrm{ns}$$

【题 4.20】

解：第一条指令LV为第一个编队。MULTSV指令因为与第一条LV指令相关，所以它们不能编在同一个编队中。但MULTSV指令和第二条LV指令之间既不存在功能部件冲突，也不存在数据相关，故可以把它们一起编到第二个编队中。ADDV指令与第二条LV指令数据相关，SV指令又与ADDV指令数据相关，所以把ADDV编为第3个编队，把SV指令编为第4个编队。即：

(1) LVV1,Rx

(2) MULTSV V2,R0,V1　　LV V3,Ry

(3) ADDV V4,V2,V3

(4) SV V4,Ry

【题 4.21】

解：当不采用向量链接技术时，可把上述5条向量指令分成以下4个编队：

(1) LV　V1,Rx

(2) MULTFV V2,F0,V1　　LV　V3,Ry

(3) ADDV V4,V2,V3

(4) SV V4,Ry

$$T_{start}=12+12+6+12, m=4$$

根据公式可知，对 n 个向量元素进行 DAXPY 表达式计算所需的时钟周期个数为：

$$T_n=\left\lceil\frac{n}{\text{MVL}}\right\rceil\times(T_{loop}+T_{start})+mn$$

$$=\left\lceil\frac{n}{64}\right\rceil\times(15+12+12+6+12)+4n=\left\lceil\frac{n}{64}\right\rceil\times 57+4n$$

如果采用向量链接技术，那么上述 5 条向量指令的编队结果如下($m=3$)：

(1) LV　V1,Rx　　MULTFV V2,F0,V1

(2) LV　V3,Ry　　ADDV V4,V2,V3

(3) SV　V4,Ry

其中，前两个编队中各自的两条向量指令都可以链接执行。根据链接的含义可知，第一个编队的启动时间为 12+7=19 个时钟周期，第二个编队的启动时间为 12+6=18 个时钟周期；第 3 个编队启动时间为 12 个时钟周期。所以采用链接技术后，对 n 个向量元素进行 DAXPY 表达式计算所需的时钟周期总数为：

$$T_n=\left\lceil\frac{n}{\text{MVL}}\right\rceil\times(T_{loop}+T_{start})+mn$$

$$=\left\lceil\frac{n}{64}\right\rceil\times(15+19+18+12)+3n=\left\lceil\frac{n}{64}\right\rceil\times 64+3n$$

第 5 章　指令级并行性及其开发——硬件方法

5.1　基本要求与难点

5.1.1　基本要求

(1) 掌握有关指令级并行的基本概念。

(2) 理解指令的静态调度和动态调度的概念，掌握动态调度的基本思想。理解精确异常和不精确异常的不同。

(3) 掌握记分牌动态调度方法的基本思想。

(4) 掌握 Tomasulo 算法的基本思想和算法。能画出在给定的情况下状态表的内容。

(5) 理解和掌握动态分支预测技术，特别是基于硬件的前瞻执行。掌握采用前瞻执行机制后指令的执行步骤发生了哪些变化。

(6) 理解超标量、超长指令字和超流水这 3 种机制的原理及特点。

(7) 了解基于静态调度的多流出技术和基于动态调度的多流出技术。

(8) 了解多流出处理器受到的限制。

5.1.2　难点

(1) 记分牌方法。

(2) Tomasulo 算法。

(3) 基于硬件的前瞻执行。

(4) 超标量、超长指令字和超流水。

5.2　知识要点

指令级并行(ILP)是指指令之间存在的一种并行性，利用它，计算机可以并行执行两条或两条以上的指令。开发 ILP 的途径有两种，一种是资源重复，重复设置多个处理部件，让它们同时执行多条指令；另一种是采用流水线技术，使指令重叠并行执行。

5.2.1　指令级并行的概念

开发 ILP 的方法可以分为两大类：主要基于硬件的动态开发方法以及基于软件的静态开发方法。两者若紧密结合，则效果更好。

流水线处理机的实际 CPI 为：

$$\text{CPI}_{\text{流水线}} = \text{CPI}_{\text{理想}} + \text{停顿}_{\text{结构冲突}} + \text{停顿}_{\text{数据冲突}} + \text{停顿}_{\text{控制冲突}}$$

其中，理想 CPI 是衡量流水线最高性能的一个指标。通过减少该式右边的各项，我们就能减少总的 CPI，从而提高 IPC(Instructions Per Cycle)。

如果一串连续的代码除了入口和出口以外，没有其他的分支指令和转入点，则称之为**一个基本程序块**。在基本程序块中能开发出的并行性是很有限的，为了明显地提高性能，必须跨越多个基本块开发 ILP。

最简单和最常用的开发 ILP 的方法，是开发循环的不同迭代之间存在的并行性。这种并行性称为**循环级并行性**。在第 6 章，将讨论如何把这种循环级并行性转换为 ILP。

5.2.2　相关与指令级并行

如果两条指令相关，则它们就不能并行执行，或只能部分重叠执行。

流水线冲突是指对于具体的流水线来说，由于相关的存在，使得指令流中的下一条指令不能在指定的时钟周期执行。流水线冲突有 3 种类型：结构冲突，数据冲突，控制冲突。结构冲突是因硬件资源冲突造成的，数据冲突是由数据相关和名相关造成的，控制冲突是由控制相关造成的。

相关是程序固有的一种属性，它反映了程序中指令之间的相互依赖关系。而具体的一次相关是否会导致实际冲突的发生以及该冲突会带来多长的停顿，则是流水线的属性。

可以从以下两个方面来解决相关问题。

(1) 保持相关，但避免发生冲突。

(2) 进行代码变换，消除相关。

指令调度是一种用来避免冲突的主要方法，它并不改变相关。

程序顺序是指：由原来程序确定的在完全串行方式下指令的执行顺序。并不需要在所有存在相关的地方都保持程序顺序。后面要介绍的各种软硬件技术的目标是尽可能地开发并行性，只有在可能会导致错误的情况下，才保持程序顺序。

对于提高性能来说，控制相关本身并不是一个主要的限制。它并不是一个必须严格保持的关键属性。为了保证程序执行的正确性，必须保持的最关键的两个属性是：**数据流**和**异常行为**。

保持异常行为是指：无论怎么改变指令的执行顺序，都不能改变程序中异常的发生情况。即原来程序中是怎么发生的，改变执行顺序后还是怎么发生。这个条件经常被弱化为：指令执行顺序的改变不能导致程序中发生新的异常。

数据流是指数据值从其产生者指令到其消费者指令的实际流动。分支指令使得数据流具有动态性，因为一条指令有可能数据相关于多条先前的指令。分支指令的执行结果决定

了哪条指令真正是所需数据的产生者。

后面将讨论的**前瞻执行**不仅能解决异常问题，而且能够使我们在保持数据流的情况下，减少控制相关对开发ILP的影响。

5.2.3　指令的动态调度

静态调度的流水线依靠编译器对代码进行静态调度，以减少相关和冲突。之所以称之为**静态调度**，是因为它不是在程序执行的过程中而是在编译期间进行代码调度和优化的。静态调度通过把相关的指令拉开"距离"来减少可能产生的停顿。

动态调度能在保持数据流和异常行为的情况下，通过硬件对指令执行顺序进行重新安排，减少数据相关导致的停顿。动态调度有许多优点：①能够处理一些编译时情况不明的相关(比如涉及存储器访问的相关)，并简化了编译器；②能够使本来是面向某一流水线优化编译的代码在其他动态调度的流水线上也能高效地执行。当然，动态调度的这些优点是以硬件复杂性的显著增加为代价的。

1. 动态调度的基本思想

第3章中讨论的5段流水线有一个很大的局限性，即其指令是按序流出和按序执行的。如果相近的指令存在相关，就很可能会导致冲突，引起停顿。这样其后面所有的指令也都停止了前进。

为了解决这个问题，我们将前述5段流水线的译码(ID)段细分为以下两个段。

(1) 流出：指令译码，并检查是否存在结构冲突。如果不存在结构冲突，就将指令流出。

(2) 读操作数：等待数据冲突消失(如果有)，然后读操作数，并立即开始执行。

可以看出，这样修改后指令的流出仍然是按序流出。但是，它们在读操作数段可能停顿或互相跨越，因而进入执行段时就可能已经是乱序的了，即**乱序执行**。指令的完成也是**乱序完成的**。

指令乱序完成大大增加了异常处理的复杂度。动态调度的处理机是这样来保持正确的异常行为的：对于一条会产生异常的指令来说，只有当处理机确切地知道该指令将被执行时，才允许它产生异常。

即使保持了正确的异常行为，动态调度处理机仍可能发生不精确异常。所谓**不精确异常**是指：当执行指令i导致发生异常时，处理机的现场(状态)与严格按程序顺序执行时指令i的现场不同。反之，如果发生异常时，处理机的现场跟严格按程序顺序执行时指令i的现场相同，就称为**精确异常**。不精确异常使得在异常处理后难以接着继续执行程序。

之所以会发生不精确的异常，是因为当发生异常(设为指令i)时：①流水线可能已经执行完按程序顺序是位于指令i之后的指令；②流水线可能还没完成按程序顺序是指令i之前的指令。

记分牌方法和Tomasulo算法是两种比较典型的动态调度算法。下面先简单介绍记分牌方法，然后详细论述Tomasulo算法。许多现代处理机都采用了Tomasulo算法或其变形。

2. 记分牌动态调度方法

1）基本思想

记分牌方法这一名称起源于最早采用此功能的 CDC 6600 计算机中的记分牌。该机器用一个称为记分牌的硬件实现了对指令的动态调度。它把前述简单流水线中的译码段 ID 分解成了两个段：流出和读操作数，以避免当某条指令在 ID 段被停顿时挡住其后面无关指令的流动。

记分牌的目标是在没有结构冲突时，尽可能早地执行没有数据冲突的指令，实现每个时钟周期执行一条指令。如果某条指令被暂停，而后面的指令与流水线中正在执行或被暂停的指令都不相关，那么这些指令可以跨越它，继续流出和执行下去。记分牌全面负责和管理这些指令的流出和执行，当然也包括检测所有的冲突。

每条指令都要经过记分牌。指令流出时，记分牌在表中记录相关的信息，并决定什么时候该指令可以读出操作数和开始执行。如果确定该指令不能马上执行，记分牌在后面就会监视硬件中信息的每一个变化，一旦所需的操作数就绪，就立即启动该指令的执行。

在采用了记分牌的流水线中。每条指令的执行过程分为 4 段：流出、读操作数、执行和写结果。由于我们主要考虑浮点操作，运算在浮点寄存器之间进行，因此不涉及存储器访问段。

（1）流出

如果当前流出指令所需的功能部件空闲，并且所有其他正在执行的指令的目的寄存器与该指令的不同，记分牌就向功能部件流出该指令，并修改记分牌内部的记录表。如果存在结构相关或 WAW 冲突，该指令就不流出。在这里就解决了 WAW 冲突。

（2）读操作数

记分牌监测源操作数的可用性，一旦数据可用，它就通知功能部件从寄存器中读出源操作数并开始执行。这一步动态地解决了 RAW 冲突，并可能导致指令乱序执行。

（3）执行

取到操作数后，功能部件开始执行。当结果产生后，就通知记分牌它已经完成执行。这一步相当于前述标准流水线中的执行段（EX）。在浮点流水线中，这一段可能要占用多个时钟周期。

（4）写结果

记分牌一旦知道执行部件完成了执行，就检测是否存在 WAR 冲突。如果不存在，或者已有的 WAR 冲突已消失，记分牌就通知功能部件把结果写入目的寄存器，并释放该指令使用的所有资源。这一步对应于前述标准流水线中的写回段（WB）。

记分牌中记录的信息由以下三部分构成。

① 指令状态表：记录正在执行的各条指令已经进入到了哪一段。

② 功能部件状态表：记录各个功能部件的状态。每个功能部件有一项，每一项由 9 个字段组成。

③ 结果寄存器状态表 Result：每个寄存器在该表中有一项，用于指出哪个功能部件（编号）将把结果写入该寄存器。

结果寄存器状态表中的字段与每个寄存器一一对应，它记录了当前机器状态下将把结

果写入该寄存器的功能部件的名称。

2）具体算法(略)

3. Tomasulo 算法

1）基本思想

Tomasulo 算法是由 Robert Tomasulo 发明的，因而以他的名字命名。首先采用这种方法的是 IBM 360/91 机器中的浮点部件。尽管许多现代处理机采用了这种方法的各种变形，但其核心思想都是：①记录和检测指令相关，操作数一旦就绪就立即执行，把发生 RAW 冲突的可能性减小到最少；②通过寄存器换名来消除 WAR 冲突和 WAW 冲突。

寄存器换名可以消除 WAR 冲突和 WAW 冲突。

下面在 MIPS 指令集的情况下来介绍 Tomasulo 算法，我们重点关心的是浮点部件以及 load/store 部件。

在 Tomasulo 算法中，寄存器换名是通过保留站和流出逻辑来共同完成的。当指令流出时，如果其操作数还没有计算出来，则将该指令中相应的寄存器号换名为将产生这个操作数的保留站的标识。所以，指令流出到保留站后，其操作数寄存器号或者换成了数据本身(若已就绪)，或者换成了保留站的标识，不再与寄存器有关系。这样后面指令对该寄存器的写入操作就不可能产生 WAR 冲突了。

在详细介绍 Tomasulo 算法之前，先来看一下指令执行的步骤。这里只需要以下三步。

(1) 流出

从指令队列的头部取一条指令。如果该指令的操作所要求的保留站有空闲的，就把该指令送到该保留站(设为 r)。并且，如果其操作数在寄存器中已经就绪，就将这些操作数送入保留站 r。如果其操作数还没有就绪，就把将产生该操作数的保留站的标识送入保留站 r。这样，一旦被记录的保留站完成计算，它将直接把数据送给保留站 r。这一步实际上是进行了寄存器换名(换成保留站的标识)和对操作数进行缓冲，消除了 WAR 冲突。另外，还要完成对目的寄存器的预约工作，将其设置为接收保留站 r 的结果。这实际上相当于提前完成了写操作(预约)。由于指令是按程序顺序流出的，当出现多条指令写同一个结果寄存器时，最后留下的预约结果肯定是最后一条指令的，就是说消除了 WAW 冲突。

当然，如果没有空闲的保留站，指令就不能流出。这是发生了结构冲突。

(2) 执行

如果某个操作数还没有被计算出来，本保留站将监视 CDB，等待所需的计算结果。一旦那个结果产生，它将被放到 CDB 上，本保留站将立即获得该数据。当两个操作数都就绪后，本保留站就用相应的功能部件开始执行指令规定的操作。这里是等到所有操作数都备齐后才开始执行指令。也就是说是靠推迟执行的方法解决 RAW 冲突。由于结果数据是从其产生的部件(保留站)直接送到需要它的地方，所以这已经是最大限度地减少了 RAW 冲突的影响。

load 和 store 指令的执行需要两个步骤：计算有效地址(要等到基地址寄存器就绪)和把有效地址放入 load 或 store 缓冲器。load 缓冲器中的 load 指令的执行条件是存储器部件就绪。而 store 缓冲器中的 store 指令在执行前必须等到要存入存储器的数据到达。通过按顺序进行有效地址计算来保证程序顺序，这有助于避免访问存储器的冲突。

(3) 写结果

功能部件计算完毕后，就将计算结果放到 CDB 上，所有等待该计算结果的寄存器和保留站(包括 store 缓冲器)都同时从 CDB 上获得所需要的数据。store 指令在这一步完成对存储器的写入：当写入地址和数据都备齐时，将它们送给存储器部件，store 指令完成。

保留站、寄存器组和 load/store 缓冲器都包含附加标志信息，用于检测和消除冲突。不同部件的附加信息略有不同。标识字段实际上就是用于换名的一组虚拟寄存器的名称(编号)。

与在 Tomasulo 算法之前提出的其他更简单的动态调度方法相比，Tomasulo 算法具有以下两个主要的优点。

① 冲突检测逻辑和指令执行控制是分布的(通过保留站和 CDB 实现)。

每个功能部件的保留站中的信息决定了什么时候指令可以在该功能部件开始执行。如果有多条指令已经获得了一个操作数，并同时在等待同一运算结果(作为另一个操作数)，那么这个结果一产生，就可以通过 CDB 同时播送给所有这些指令，使它们可以同时执行。

② 消除了 WAW 冲突和 WAR 冲突导致的停顿。

这是通过使用保留站进行寄存器换名，并且在操作数一旦就绪就将之放入保留站来实现的。

2) 具体算法

下面给出 Tomasulo 算法中指令进入各阶段的条件以及在各阶段进行的操作和状态表内容修改。其中，各符号的意义如下。

r：分配给当前指令的保留站或者缓冲器单元(编号)；

rd：目的寄存器编号；

rs、rt：操作数寄存器编号；

imm：按符号位扩展后的立即数；

RS：保留站；

result：浮点部件或 load 缓冲器返回的结果；

Qi：寄存器状态表；

Regs[]：寄存器组；

Op：当前指令的操作码。

对于 load 指令来说，rt 是保存所取数据的寄存器号；对于 store 指令来说，rt 是保存所要存储的数据的寄存器号。与 rs 对应的保留站字段是 Vj、Qj；与 rt 对应的保留站字段是 Vk、Qk。

(1) 指令流出

① 浮点运算指令

```
进入条件：有空闲保留站(设为 r)
操作和状态表内容修改：
  if  (Qi[rs] ≠ 0)             //检测第一操作数是否就绪
     {RS[r].Qj ← Qi[rs]        //第一操作数没有就绪，进行寄存器换名，即把将产生该操作数的保
                               //留站的编号放入当前保留站的 Qj。该编号是一个大于 0 的整数
  else {RS[r].Vj ← Regs[rs];  //第一操作数就绪。把寄存器 rs 中的操作数取到当前保留站的 Vj
     RS[r].Qj ← 0};            //置 Qj 为 0，表示当前保留站的 Vj 中的操作数就绪
```

```
  if  (Qi[rt] ≠ 0)          //检测第二操作数是否就绪
     {RS[r].Qk ← Qi[rt]}    //第二操作数没有就绪,进行寄存器换名,即把将产生该操作数的保
                            //留站的编号放入当前保留站的Qk。该编号是一个大于0的整数
  else {RS[r].Vk← Regs[rt]; //第二操作数就绪。把寄存器rt中的操作数取到当前保留站的Vk
     RS[r].Qk← 0};          //置Qk为0,表示当前保留站的Vk中的操作数就绪
  RS[r].Busy ← yes;         //置当前保留站为"忙"
  RS[r].Op ← Op;            //设置操作码
  Qi[rd] ← r;               //把当前保留站的编号r放入rd所对应的寄存器状态表项,
                            //以便rd将来接收结果
```

② load 和 store 指令

进入条件：缓冲器有空闲单元(设为r)
操作和状态表内容修改：

```
  if  (Qi[rs] ≠ 0)        //检测第一操作数是否就绪。
     {RS[r].Qj ← Qi[rs]}  //第一操作数没有就绪,进行寄存器换名,即把将产生该操作数的保
                          //留站的编号存入当前缓冲器单元的Qj
  else
    {RS[r].Vj ← Regs[rs]; //第一操作数就绪,把寄存器rs中的操作数取到当前缓冲器单元
                          //的Vj
    RS[r].Qj ← 0};        //置Qj为0,表示当前缓冲器单元的Vj中的操作数就绪
  RS[r].Busy ← yes;       //置当前缓冲器单元为"忙"
  RS[r].A ← Imm;          //把按符号位扩展后的偏移量放入当前缓冲器单元的A
对于load指令：
  Qi[rt] ← r;             //把当前缓冲器单元的编号r放入load指令的目的寄存器rt所对应
                          //的寄存器状态表项,以便rt将来接收所取的数据
对于store指令：
  if  (Qi[rt] ≠0)         //检测要存储的数据是否就绪
     {RS[r].Qk ← Qi[rt]}  //该数据尚未就绪,进行寄存器换名,即把将产生该数据的保留站的
                          //编号放入当前缓冲器单元的Qk
  else
    {RS[r].Vk ← Regs[rt]; //该数据就绪,把它从寄存器rt取到store缓冲器单元的Vk
     RS[r].Qk ← 0};       //置Qk为0,表示当前缓冲器单元的Vk中的数据就绪
```

(2) 执行

① 浮点操作指令

进入条件：(RS[r].Qj = 0)且(RS[r].Qk = 0);　　//两个源操作数就绪
操作和状态表内容修改：进行计算,产生结果

② load/store 指令

进入条件：(RS[r].Qj = 0)且r成为load/store缓冲队列的头部
操作和状态表内容修改：

```
  RS[r].A ← RS[r].Vj + RS[r].A;        //计算有效地址
对于load指令,在完成有效地址计算后,还要进行:
  从Mem[RS[r].A]读取数据;              //从存储器中读取数据
```

(3) 写结果

① 浮点运算指令和 load 指令

进入条件：保留站r执行结束,且CDB就绪

```
操作和状态表内容修改:
   ∀x  (if(Qi[x] = r)             //对于任何一个正在等该结果的浮点寄存器 x,
     { Regs[x] ← result;          //向该寄存器写入结果
     Qi[x] ← 0};                  //把该寄存器的状态置为数据就绪
   ∀x  (if(RS[x].Qj = r)          //对于任何一个正在等该结果作为第一操作数的保留站 x,
     {RS[x].Vj ← result;          //向该保留站的 Vj 写入结果
     RS[x].Qj ← 0});              //置 Qj 为 0,表示该保留站的 Vj 中的操作数就绪
   ∀x  (if(RS[x].Qk = r)          //对于任何一个正在等该结果作为第二操作数的保留站 x,
     {RS[x].Vk ← result;          //向该保留站的 Vk 写入结果
     RS[x].Qk ← 0});              //置 Qk 为 0,表示该保留站的 Vk 中的操作数就绪
  RS[r].Busy ← no;                //释放当前保留站,将之置为空闲状态
```

② store 指令

```
进入条件: 保留站 r 执行结束,且 RS[r].Qk = 0   //要存储的数据已经就绪
操作和状态表内容修改:
  Mem[RS[r].A]← RS[r].Vk          //数据写入存储器,地址由 store 缓冲器单元的 A 字段给出
RS[r].Busy← no;                   //释放当前缓冲器单元,将之置为空闲状态
```

如果能够准确地预测分支,采用 Tomasulo 算法将获得很高的性能。这种方法的主要缺点是其复杂性,实现它需要大量的硬件。所以在单流出的流水线中,采用 Tomasulo 算法所带来的好处与所花的代价相比不一定值得。但是,对于多流出的处理机来说,随着流出能力的提高,寄存器换名以及动态调度技术就变得越来越重要了。特别是,Tomasulo 算法还是硬件前瞻执行的基础,因此该算法得到了广泛的应用。

5.2.4　动态分支预测技术

开发的 ILP 越多,控制相关的制约就越大,就要求分支预测有更高的准确度。本节中介绍的方法对于每个时钟周期流出多条指令的处理机来说是非常重要的,这是因为:①在 n 流出(每个时钟周期流出 n 条指令)的处理机中,遇到分支指令的可能性增加了 n 倍。要给处理器连续提供指令,就需要预测分支的结果。②Amdahl 定律告诉我们,机器的 CPI 越小,控制停顿的相对影响就越大。

采用这些动态分支预测技术的目的有两个:预测分支是否成功和尽快找到分支目标地址(或指令),从而避免控制相关造成流水线停顿。在这些方法中,需要解决以下关键问题:①如何记录分支的历史信息;②如何根据这些信息来预测分支的去向,甚至提前取出分支目标处的指令。

1. 采用分支历史表 BHT

分支历史表(Branch History Table,BHT)法是最简单的动态分支预测方法。它用 BHT 来记录相关分支指令的“历史”,并据此进行预测。这个“历史”是指最近一次或几次的执行情况是成功还是失败。常采用两位二进制位来记录历史。有研究结果表明,两位分支预测的性能与多位(两位以上)分支预测的性能差不多。因而大多数处理机是采用两位分支预测。

研究结果表明,对于 SPEC89 测试程序来说,大小为 4K 的 BHT 的预测准确率为

82%～99%，并且与大小为无穷大的BHT的准确率相近。所以一般来说，采用4K的BHT就可以了。

2. 采用分支目标缓冲器BTB

在多流出的处理机中，只准确地预测分支还不够，还要能够快速地提供足够的指令流。许多现代的处理器都要求每个时钟周期能提供4～8条指令。这需要尽早知道分支是否成功，尽早知道分支目标地址，尽早获得分支目标指令。

对于前述5段流水线来说，BHT方法是在ID段对BHT进行访问，所以在ID段的末尾，能够获得分支目标地址(在ID段计算出)、顺序下一条指令地址以及预测的结果。如果能再提前一拍，即在IF段就知道这些信息，那么分支开销就可以减少为0。BTB能够实现这一点。BTB是Branch Target Buffer的缩写，其中文名称是**分支目标缓冲器**。BTB有时也称为分支目标Cache。

BTB可以看成是用专门的硬件实现的一张表格。表格中的每一项至少有两个字段：①执行过的成功分支指令的地址；②预测的分支目标地址。以第一个字段作为该表的匹配标识。在每次取指令的同时，用该指令的地址与BTB中所有项目的第一个字段进行比较。如果有匹配的，我们就知道该指令是分支指令且上一次执行是分支成功，据此我们预测这次执行也将分支成功，其分支目标地址由匹配项的第二个字段给出。如果没有匹配的，就把当前指令当作普通的指令(即不是分支指令)来执行。

当采用BTB后，如果当前指令的地址与BTB中的第一字段匹配，则从分支目标处开始取指令。如果预测正确，则不会有任何分支延迟。如果预测错误或者在BTB中没有匹配的项，则至少会有两个时钟周期的开销。

BTB的另外两种形式是在分支目标缓冲器中增加一个字段来存放BHT表，或者增加一个字段来存放若干条分支目标处的指令。后者可以一次性提供分支目标处的多条指令，这对于多流出处理器来说是很有必要的。

3. 基于硬件的前瞻执行

对于多流出的处理机来说，控制相关已经成了开发更多ILP的一个主要障碍。**前瞻执行**能很好地解决控制相关的问题，它对分支指令的结果进行猜测，然后按这个猜测结果继续取指、流出和执行后续的指令。只是指令执行的结果不是写回到寄存器或存储器，而是放到一个称为**ROB**(ReOrder Buffer)的缓冲器中。等到相应的指令得到"确认"(即确实是应该执行的)后，才将结果写入寄存器或存储器。之所以要这样，是为了在猜测错误的情况下能够恢复原来的现场

基于硬件的前瞻执行是把3种思想结合在了一起。

(1) 动态分支预测。用来选择后续执行的指令。

(2) 在控制相关的结果尚未出来之前，前瞻地执行后续指令。

(3) 用动态调度对基本块的各种组合进行跨基本块的调度。

对Tomasulo算法加以扩充，就可以支持前瞻执行。当然，硬件也需要做相应的扩展。在Tomasulo算法中，写结果和指令完成是一起在"写结果"段完成的。现在我们把该段分为"写结果"和"指令确认"两个段。"**写结果**"段是把前瞻执行的结果写到ROB中，并通过

CDB 在指令之间传送结果,供需要用到这些结果的指令使用。**"指令确认"**段是在分支指令的结果出来后,对相应指令的前瞻执行给予确认,把在 ROB 中的结果写到寄存器或存储器(如果前面所做的猜测是对的)。如果发现前面对分支结果的猜测是错误的,那就不予以确认,并从那条分支指令的另一条路径开始重新执行。

前瞻执行允许指令乱序执行,但要求按程序顺序确认。并且在指令被确认之前,不允许它进行不可恢复的操作,如更新机器状态或发生异常。

支持前瞻执行的浮点部件的结构与基于 Tomasulo 算法的 MIPS 处理器浮点部件的基本结构相比,主要是增加了一个 ROB 缓冲器。ROB 是为实现前瞻执行而设置的,它在指令操作完成后到指令被确认这段时间,为指令保存数据。正在前瞻执行的指令之间也是通过 ROB 传送结果的,因为前瞻执行的指令在被确认前是不能写寄存器的。

ROB 中的每一项由以下 4 个字段组成。

(1) 指令类型:指出该指令是分支指令、store 指令或寄存器操作指令。

(2) 目的地址:给出指令执行结果应写入的目的寄存器号(如果是 load 和 ALU 指令)或存储器单元的地址(如果是 store 指令)。

(3) 数据值字段:用来保存指令前瞻执行的结果,直到指令得到确认。

(4) 就绪字段:指出指令是否已经完成执行并且数据已就绪。

在前瞻执行机制中,Tomasulo 算法中保留站的换名功能是由 ROB 来完成的。但在指令流出到开始执行期间,仍然需要有地方来存放运算操作码和操作数。这个功能仍由保留站来完成。由于每条指令在被确认前,在 ROB 中都有相应的一项,所以对于执行结果我们是用 ROB 项的编号作为标识,而不像 Tomasulo 算法那样是用保留站的编号。这就要求在保留站中记录分配给该指令的 ROB 项编号。

采用前瞻执行机制后,指令的执行步骤如下。

1) 流出

从浮点指令队列的头部取一条指令,如果有空闲的保留站(设为 r)且有空闲的 ROB 项(设为 b),就流出该指令,并把相应的信息放入保留站 r 和 ROB 项 b。即:如果该指令需要的操作数已经在寄存器或 ROB 中就绪,就把它(们)送入保留站 r 中。修改 r 和 b 的控制字段,表示它们已经被占用。ROB 项 b 的编号也要放入保留站 r,以便当该保留站的执行结果被放到 CDB 上时可以用它作为标识。如果保留站或 ROB 全满,便停止流出指令,直到它们都有空闲的项。

2) 执行

如果有操作数尚未就绪,就等待,并不断地监测 CDB。这一步检测 RAW 冲突。当两个操作数都已在保留站中就绪后,就可以执行该指令的操作。load 指令的操作还是分两步完成(有效地址计算和读取数据),store 指令在这一步只进行有效地址计算。

3) 写结果

当结果产生后,将该结果连同本指令在流出段所分配到的 ROB 项的编号放到 CDB 上,经 CDB 写到 ROB 以及所有等待该结果的保留站。然后释放产生该结果的保留站。store 指令的操作有些特殊(与 Tomasulo 算法不同):如果要写入存储器的数据已经就绪,就把该数据写入分配给该 store 指令的 ROB 项。否则,就监测 CDB,直到那个数据在 CDB 上播送出来,才将其写入分配给该 store 指令的 ROB 项。

4）确认

这一阶段对分支指令、store指令以及其他指令的处理不同。

(1) 对于除分支指令和store指令以外的指令来说，当该指令到达ROB队列的头部而且其结果已经就绪时，就把该结果写入该指令的目的寄存器，并从ROB中删除该指令。

(2) 对store指令的处理与(1)类似，只是它是把结果写入存储器。

(3) 当预测错误的分支指令到达ROB队列的头部时，就表示是错误的前瞻执行。这时要清空ROB，并从分支指令的另一个分支重新开始执行。

(4) 当预测正确的分支指令到达ROB队列的头部时，该指令执行完毕。

一旦指令得到确认，就释放它所占用的ROB项。当ROB满时，就停止指令的流出，直到有空闲项被释放出来。

由于前瞻执行通过ROB实现了指令的顺序完成，所以它不仅能够进行前瞻执行，而且能够实现精确异常。

前瞻执行的主要缺点是：所需的硬件太复杂。与Tomasulo算法相比，在控制方面复杂多了，因而在控制逻辑硬件方面增加了许多。

5.2.5 多指令流出技术

在单流出的情况下，CPI不可能小于1。如果想进一步提高性能，使CPI小于1，就必须采用**多流出技术**，在每个时钟周期流出多条指令。

多流出处理机有两种基本风格：超标量和超长指令字(Very Long Instruction Word, VLIW)。**超标量**在每个时钟周期流出的指令条数不固定，依代码的具体情况而定，不过有个上限。如果这个上限为n，就称该处理机为**n-流出**。对于超标量处理机，既可以通过编译器进行静态调度，也可以基于Tomasulo算法进行动态调度。静态调度的超标量处理机一般采用按序执行，而动态调度的处理机一般采用乱序执行。

与超标量处理机不同，超长指令字VLIW处理机在每个时钟周期流出的指令条数是固定的，这些指令构成一条长指令或者一个指令包。在这个指令包中，指令之间的并行性是通过指令显式地表示出来的。这种处理机的指令调度由编译器静态完成。

与VLIW处理机相比，超标量处理机有两个优点。

(1) 超标量结构对程序员是透明的，处理机能自己检测下一条指令是否能流出，不需要由编译器或专门的变换程序对程序中的指令进行重新排列。

(2) 即使是没有经过编译器针对超标量结构进行调度优化的代码或是旧的编译器生成的代码也可以运行，当然运行的效果不会很好。要想达到很好的效果，方法之一就是使用动态超标量调度技术。

1. 基于静态调度的多流出技术

在静态调度的超标量处理机中，指令按序流出。所有的冲突检测都在流出时进行，由硬件检测当前流出的指令之间是否存在冲突以及它们与正在执行的指令之间是否有冲突。如果在当前流出的指令序列中，某条指令存在冲突，那么就只流出该指令之前的指令。

考虑一个4流出的静态调度超标量处理机。在取指令阶段，流水线将从取指令部件收

到1～4条指令，流出部件通过检测冲突来确定这些指令是全部流出还是部分流出。由于这些检测比较复杂，难以在一个时钟周期内完成，所以许多静态调度的超标量处理机都是将其分成多个流水段，按流水方式工作。动态调度的超标量处理机中也是如此。

如果MIPS处理机按超标量方式工作，结果将会怎样？我们假设每个时钟周期可以流出两条指令："1条整数型指令+1条浮点操作指令"，其中把load指令、store指令、分支指令也归类为整数型指令。与任意的双流出相比，把整数指令和浮点指令结合流出是简单了不少，对硬件的要求也没那么高。

为了实现每个时钟周期流出两条指令，显然是要能够同时取两条指令(64位)，也要能同时译码两条指令(64位)。对指令的处理包括以下步骤：①从Cache中取两条指令；②确定哪几条指令可以流出(0～2条指令)；③把它们发送到相应的功能部件。取两条指令还比较容易实现，若要取更多的指令，所要进行的处理就复杂多了。

对于上述简单的超标量处理机来说，冲突检测还比较简单，因为"1条整数型指令+1条浮点指令"的流出方式消除了大多数流出包内的冲突。主要的难点出现在当整数型指令是一条浮点load、store或move指令的情况。这时有可能会出现争用浮点寄存器端口或者产生新的RAW冲突。采用"1条整数型指令+1条浮点指令"并行流出的方式，需要增加的硬件很少。这是因为整数指令和浮点指令使用不同的寄存器组和不同的功能部件。

为了能有效地利用超标量处理机所具有的并行性，需要采用更有效的编译技术或者硬件调度技术。如果不采用这些技术，超标量技术所能带来的性能上的提高可能很有限。

2. 基于动态调度的多流出技术

在多流出处理机中，动态调度技术是提高性能的一种方法。动态调度不仅拥有能解决数据冲突和提高性能的典型优点，而且有可能克服指令流出所受的限制。尽管从硬件的角度来看，在每个时钟周期最多只能启动一个整数操作和一个浮点操作的执行，但动态调度可以使得在指令流出时不受这个限制，至少在保留站被全部占用之前是如此。

假设我们要对Tomasulo算法进行扩展，使之能支持双流出超标量流水线，但又不想乱序地向保留站流出指令，因为这会破坏程序语义。为了充分利用动态调度的好处，我们也许应该去掉每个时钟周期只能流出"1条整数型指令+1条浮点指令"的限制，但这会大大增加指令流出的硬件复杂度。

在采用动态调度的处理机中，有两种不同的方法可以用来实现多流出。它们都是建立在这样一个观点之上的：动态调度关键在于对保留站的分配和对流水线控制表格的修改。一种方法是在半个时钟周期内完成流出步骤，这样一个时钟周期就能处理两条指令。另一种方法是设置一次能同时处理两条指令的逻辑电路。现代的流出4条或4条以上指令的超标量处理机经常是综合采用这两种方法，即：不仅采用流水，而且还把流出电路加宽。

3. 超长指令字技术

下面只做简单的介绍。更详细的论述见第6章。

超长指令字(Very Long Instruction Word，VLIW)技术是另一种多指令流出技术。与超标量不同，它在指令流出时不需要进行复杂的冲突检测，而是依靠编译器全部安排好了。在编译时，编译器找出指令之间潜在的并行性，并通过指令调度把可能出现的数据冲突减少

到最少，最后把能并行执行的多条指令组装成一条很长的指令。这种指令字经常是一百多位到几百位，超长指令字因此得名。

在VLIW处理机中一般设置有多个功能部件。相应地，指令字也被分割成一些字段，每个字段称为一个操作槽，直接独立地控制一个功能部件。为了使功能部件充分忙碌，程序指令序列中应有足够的并行性，从而尽量填满每个操作槽。这种并行性是完全依靠编译器来挖掘的。它不需要超标量处理机中用于指令流出控制的硬件，因而控制硬件比较简单。特别是当流出宽度增加时，VLIW技术的优点更加明显。

当然，VLIW也存在一些问题，包括：

(1) 程序代码长度增加了；

(2) 采用了锁步机制；

(3) 机器代码的不兼容性。

4. 多流出处理器受到的限制

指令多流出处理器的流出能力主要受以下3个方面的影响。

(1) 程序所固有的指令级并行性；

(2) 硬件实现上的困难；

(3) 超标量和超长指令字处理器固有的技术限制。

其中，第一个限制是最简单的也是最根本的因素。对于流水线处理器，需要有大量可并行执行的操作才能避免流水线出现停顿。如果浮点流水线的延迟为5个时钟周期，要使该浮点流水线不停顿，就必须有5条无相关的浮点指令。通常情况下，所需要的无相关指令数等于流水线的深度乘以可以同时工作的功能部件数。

第二个限制，是多流出的处理器需要大量的硬件资源。因为每个时钟周期不仅要流出多条指令，而且要执行它们。随着每个时钟周期流出指令数的增加，所需的硬件成正比例地增长，所需的存储器带宽和寄存器带宽也大大增加了，这样的带宽要求必然导致大量增加硅片面积，加大面积就导致时钟频率下降、功耗增加、可靠性降低等一系列问题。

如果要使流出指令的数目增加，就需要进一步增加更多的存储器端口。多端口、层次化的存储系统带来的系统复杂性和访问延迟，可能是指令多流出技术所面临的最严重的硬件的限制。

多指令流出所需的硬件量随实现方法的不同有很大的差别。一个极端是动态调度的超标量处理器，无论采用计分牌技术还是Tomasulo算法，都需要大量的硬件，而且动态调度也大大增加了设计的复杂性，使提高时钟频率更加困难。另一个极端是VLIW处理器，指令的流出和调度仅需要很少甚至不需要额外的硬件，因为这些工作全都由编译器进行。这两种极端之间是现存的多数超标量处理器，它们将编译器的静态调度和硬件的动态调度机制结合起来，共同决定可同时并行流出的指令数。

5. 超流水线处理机

在第3章介绍的流水处理机中，是把一条指令的执行过程分解为取指令、译码、执行、访存、写结果5个流水段。如果把其中的每个流水段进一步细分，例如，分解为两个延迟时间更短的流水段，则一条指令的执行过程就要经过10个流水段。这样，在一个时钟周期内，取

指令、译码、执行、访存、写结果等各段都在处理各自的两条指令。这种在一个时钟周期内能够分时流出多条指令的处理机称为**超流水线处理机**。

与超标量处理机不同，超流水线处理机只需增加少量硬件，是通过各部分硬件的充分重叠工作来提高性能的。超标量处理机采用的是空间并行性，而超流水线处理机采用的是时间并行性。

对于一台每个时钟周期能流出 n 条指令的超流水线计算机来说，这 n 条指令不是同时流出的，而是每隔 $1/n$ 个时钟周期流出一条指令。因此，实际上该流水线的工作周期是系统时钟周期的 $1/n$。

在有的资料中，把指令流水线级数为 8 或 8 以上的流水线处理机称为超流水线处理机。

习　　题

1. 概念题

【题 5.1】　解释下列名词

指令级并行	IPC	循环级并行性	指令的动态调度
指令的静态调度	不精确异常	精确异常	CDB
BHT	分支目标缓冲	ROB	超标量
超流水	超长指令字		

2. 填空题

【题 5.2】　开发指令级并行的方法主要有两类：基于硬件的________方法以及基于软件的________方法。

【题 5.3】　如果一串连续的代码除了入口和出口以外，没有其他的分支指令和转入点，则称之为一个________。

【题 5.4】　说出两种比较典型的动态调度算法：________和________。

【题 5.5】　要扩充 Tomasulo 算法支持前瞻执行，需将 Tomasulo 算法中的“写结果”段分为________和________两个段。

【题 5.6】　前瞻执行允许指令________执行，但要求按________确认。

【题 5.7】　Tomasulo 算法中换名功能是由________来完成；而在前瞻执行机制中，换名功能是由________来完成的。

【题 5.8】　静态指令调度技术是优化的________来完成，其基本思想是重排指令序列，拉开具有________的有关指令间的距离。

【题 5.9】　动态分支预测的依据是从________指令过去的行为来预测它将来的行为，即根据近期转移是否成功的________记录，来预测下一次转移的________。

【题 5.10】　多流出处理机有________和________两种基本风格。

3. 问答题

【题 5.11】　简述开发指令级并行 ILP 的两种途径。

【题5.12】 为了保证程序执行的正确性,必须保持哪两个最关键的属性?并简述其含义。

【题5.13】 指令的动态调度有何优点?

【题5.14】 记分牌算法中,记分牌中记录的信息由哪三部分构成?

【题5.15】 简述Tomasulo算法的基本思想。

【题5.16】 简述Tomasulo算法中,指令流出段所做的主要工作。

【题5.17】 与在Tomasulo算法之前提出的其他更简单的动态调度方法相比,Tomasulo算法具有哪两个主要的优点?

【题5.18】 采用动态分支预测技术的目的是什么?在所采用的方法中,需要解决哪些关键问题?

【题5.19】 给出采用分支目标缓冲器BTB后,在流水线3个阶段(IF段、ID段、EX段)所进行的相关操作。

【题5.20】 基于硬件的前瞻执行是把哪3种思想结合在了一起?

【题5.21】 ROB中的每一项由哪4个字段组成?并简述其含义。

【题5.22】 简述采用前瞻执行机制后,指令确认段所做的主要工作。

【题5.23】 与VLIW处理机相比,超标量处理机有什么优点?

【题5.24】 指令多流出处理器的流出能力主要受哪3个方面的影响?

4. 应用题

【题5.25】 有A、B、C、D 4个存储器操作数,要求完成$(A\times B)+(C+D)$的运算,原来使用的程序如下:

```
I1  LOAD  R1,M(A)        ;R1←M(A)
I2  LOAD  R2,M(B)        ;R2←M(B)
I3  MUL   R5,R1,R2       ;R5←(R1)*(R2)
I4  LOAD  R3,M(C)        ;R3←M(C)
I5  LOAD  R4,M(D)        ;R4←M(D)
I6  ADD   R2,R3,R4       ;R2←(R3)+(R4)
I7  ADD   R2,R2,R5       ;R2←(R2)+(R5)
```

现采用静态指令调度方法,请写出该程序调度后的指令序列。

【题5.26】 假定有多个加法器,不存在加法器的资源冲突。有3条连续指令组成的程序代码如下:

```
I1  ADD  R1, R2, R4      ;R1←(R2)+(R4)
I2  ADD  R2, R1, 1       ;R2←(R1)+ 1
I3  SUB  R1, R4, R5      ;R1←(R4)-(R5)
```

(1) 分析程序代码段中的数据相关。

(2) 采用何种硬件技术可解决这些数据相关?请加以说明。

【题5.27】 假设有一条长流水线,仅对条件转移指令使用分支目标缓冲。假设分支预测错误的开销为4个时钟周期,缓冲不命中的开销为3个时钟周期。假设:命中率为90%,预测精度为90%,分支频率为15%,没有分支的基本CPI为1。

(1) 求程序执行的 CPI。

(2) 相对于采用固定的两个时钟周期延迟的分支处理，哪种方法程序执行速度更快？

【题 5.28】 假设分支目标缓冲的命中率为 90%，程序中无条件转移指令的比例为 5%，没有无条件转移指令的程序 CPI 值为 1。假设分支目标缓冲中包含分支目标指令，允许无条件转移指令进入分支目标缓冲，则程序的 CPI 值为多少？假设原来的 CPI=1.1。

【题 5.29】 对于下述指令序列：

```
L.D       F6, 34(R2)
L.D       F2, 45(R3)
MUL.D     F0, F2, F4
SUB.D     F8, F2, F6
DIV.D     F10, F0, F6
ADD.D     F6, F8, F2
```

(1) 给出当第一条指令完成并写入结果时，Tomasulo 算法所用的各信息表中的内容。

(2) 假设各种操作的延迟为：

load: 1 个时钟周期
加法: 2 个时钟周期
乘法: 10 个时钟周期
除法: 40 个时钟周期

给出 MUL.D 指令准备写结果时各状态表的内容。

【题 5.30】 假设浮点功能部件的延迟时间为：加法两个时钟周期，乘法 10 个时钟周期，除法 40 个时钟周期。对于下面的代码段，在基于 Tomasulo 算法的支持前瞻执行的浮点部件的结构上，给出当指令 MUL.D 即将确认时的状态表内容。

```
L.D       F6, 34(R2)
L.D       F2, 45(R3)
MUL.D     F0, F2, F4
SUB.D     F8, F6, F2
DIV.D     F10, F0, F6
ADD.D     F6, F8, F2
```

【题 5.31】 下面的一段 MIPS 汇编程序是计算高斯消去法中的关键一步，用于完成下面公式的计算：

$$Y = a \times X + Y$$

指令的延迟如表 5.1 所示。

表 5.1　指令的延迟

产生结果的指令	使用结果的指令	延迟(时钟周期数)
浮点计算	另一个浮点计算	3
浮点计算	浮点 store(S.D)	2
浮点 load(L.D)	浮点计算	1
浮点 load(L.D)	浮点 store(S.D)	0

整数指令均为一个时钟周期完成，浮点和整数部件均采用流水。整数操作之间以及与其他所有浮点操作之间的延迟为 0，转移指令的延迟为 0。X 中的最后一个元素存放在存储器中的地址为 DONE。

```
FOO:    L.D      F2, 0(R1)
        MUT.D    F4, F2, F0
        L.D      F6, 0(R2)
        ADD.D    F6, F4, F6
        S.D      F6, 0[R2]
        DADDIU   R1, R1, #8
        DADDIU   R2, R2, #8
        DSUBIU   R3, R1, #DONE
        BNEZ     R3, FOO
```

(1) 对于标准的 MIPS 单流水线，上述循环计算一个 Y 值需要多少时间？其中有多少空转周期？

(2) 对于标准的 MIPS 单流水线，将上述循环顺序展开 4 次，不进行任何指令调度，计算一个 Y 值平均需要多少时间？加速比是多少？其加速是如何获得的？

(3) 对于标准的 MIPS 单流水线，将上述循环顺序展开 4 次，优化和调度指令，使循环处理时间达到最优，计算一个 Y 值平均需要多少时间？加速比是多少？

(4) 对于采用前瞻执行机制的 MIPS 处理器(只有一个整数部件)，当循环第二次执行到 BNEZ R3，FOO 时，写出前面所有指令的状态，包括指令使用的保留站、指令起始节拍、执行节拍和写结果节拍，并写出处理器当前的状态。

(5) 对于两路超标量的 MIPS 流水线，设有两个指令流出部件，可以流出任意组合的指令，系统中的功能部件数量不受限制。将上述循环展开 4 次，优化和调度指令，使循环处理时间达到最优。计算一个 Y 值平均需要多少时间？加速比是多少？

(6) 对于超长指令字 MIPS 处理器，将上述循环展开 4 次，优化和调度指令，使循环处理时间达到最优。计算一个 Y 值平均需要多少时间？加速比是多少？

【题 5.32】 对于两路超标量处理器，从存储器取数据有两拍附加延迟，其他操作均有 1 拍附加延迟，对于下列代码，请按要求进行指令调度。

```
LW      R4, (R5)
LW      R7, (R8)
DADD    R9, R4, R7
LD      R10, (R11)
DMUL    R12, R13, R14
DSUB    R2, R3, R1
SW      R15, (R2)
DMUL    R21, R4, R7
SW      R23, (R22)
SW      R21, (R24)
```

(1) 假设两路功能部件中同时最多只有一路可以是访问存储器的操作，同时也最多只有一路可以是运算操作，指令顺序不变。

(2) 假设两路功能部件均可以执行任何操作，指令顺序不变。

(3) 假设指令窗口足够大，指令可以乱序(out-of-order)流出，两路功能部件均可以执行任何操作。

【题 5.33】 设指令流水线由取指令、分析指令和执行指令 3 个部件构成，每个部件经过的时间为 Δt，连续流入 12 条指令。分别画出标量流水处理机以及 ILP 均为 4 的超标量处理机、超长指令字处理机、超流水处理机的时空图，并分别计算它们相对于标量流水处理机的加速比。

【题 5.34】 用一台每个时钟周期发射两条指令的超标量处理机运行下面一段程序。所有指令都要进行取指(IF)、译码(ID)、执行、写结果(WB)4 个阶段。其中，IF、ID、WB 三个阶段各为一个流水段，其延迟时间都为 10ns。在执行阶段，LOAD 操作、AND 操作各延迟 10ns，ADD 操作延迟 20ns，MUL 操作延迟 30ns。这 4 种功能部件各设置一个，它们可以并行工作。ADD 部件和 MUL 部件都采用流水结构，每一级流水线的延迟时间都是 10ns。

```
I1  LOAD  R0, M(A)      ;R0←M(A)
I2  ADD   R1, R0        ;R1←(R1)+(R0)
I3  LOAD  R2, M(B)      ;R2←M(B)
I4  MUL   R3, R4        ;R3←(R3)×(R4)
I5  AND   R4, R5        ;R4←(R4)∧(R5)
I6  ADD   R2, R5        ;R2←(R2)+(R5)
```

(1) 请列出程序代码中所有的数据相关及其相关类型。

(2) 假设所有运算型指令都在译码(ID)流水段读寄存器、在写结果(WB)流水段写寄存器，采用顺序发射顺序完成的调度方法。画出流水线的时空图；计算执行这个程序所用的时间。

【题 5.35】 对于采用了 Tomasulo 算法和多流出技术的 MIPS 流水线，考虑以下简单循环的执行。该程序把 F2 中的标量加到一个向量的每个元素上。

```
Loop:  L.D      F0, 0(R1)       //取一个数组元素放入 F0
       ADD.D    F4, F0, F2      //加上在 F2 中的标量
       S.D      F4, 0(R1)       //存结果
       DADDIU   R1, R1, # -8    //指针减 8(每个数据占 8 个字节)
       BNE      R1, R2, Loop    //若 R1 不等于 R2,表示尚未结束,转移
                                //到 Loop 继续
```

现做以下假设：

(1) 每个时钟周期能流出一条整数指令和一条浮点指令，即使它们相关也是如此。

(2) 整数 ALU 运算和地址计算共用一个整数部件；并且对于每一种浮点操作类型都有一个独立的流水化了的浮点功能部件。

(3) 指令流出和写结果各占用一个时钟周期。

(4) 具有动态分支预测部件和一个独立的计算分支条件的功能部件。

(5) 跟大多数动态调度处理器一样，写回段的存在意味着实际的指令延迟会比按序流动的简单流水线多一个时钟周期。所以，从产生结果数据的源指令到使用该结果数据的指令之间的延迟为：整数运算一个周期，load 两个周期，浮点加法运算 3 个周期。

(1) 请列出该程序前面三遍循环中各条指令的流出、开始执行和将结果写到 CDB 上的时间。

(2) 如果分支指令单流出，没有采用延迟分支，但分支预测是完美的。请列出整数部件、浮点部件、数据 Cache 以及 CDB 的资源使用情况。

【题 5.36】 在基于记分牌的 MIPS 处理器的基本结构上，运行下列代码，指令状态表的内容如表 5.2 所示。给出功能部件状态表和结果寄存器状态表中保存的信息。其中，有一个整数部件 Integer，两个乘法部件 Mult1 和 Mult2，一个加法部件 Add 和一个除法部件 Divide。

```
L. D      F6, 34(R2)
L. D      F2, 45(R3)
MULT.D    F0, F2, F4
SUB.D     F8, F6, F2
DIV.D     F10, F0, F6
ADD.D     F6, F8, F2
```

表 5.2 指令状态表

指令		指令状态表			
		流出	读操作数	执行	写结果
L.D	F6,34(R2)	√	√	√	√
L.D	F2, 45(R3)	√	√	√	
MULT.D	F0, F2, F4	√			
SUB.D	F8, F6, F2	√			
DIV.D	F10, F0, F6	√			
ADD.D	F6, F8, F2				

【题 5.37】 假设浮点流水线中各部件的延迟如下：加法需两个时钟周期、乘法需 10 个时钟周期、除法需 40 个时钟周期。运行下列代码段：

```
L. D      F6, 34(R2)
L. D      F2, 45(R3)
MULT.D    F0, F2, F4
SUB.D     F8, F6, F2
DIV.D     F10, F0, F6
ADD.D     F6, F8, F2
```

在记分牌算法中，给出 MULT. D 准备写结果之前的记分牌状态。其中，有一个整数部件 Integer，两个乘法部件 Mult1 和 Mult2，一个加法部件 Add 和一个除法部件 Divide。

题　　解

1. 概念题

【题 5.1】 解释下列名词

指令级并行 —— 简称 ILP。是指指令之间存在的一种并行性，利用它，计算机可以并行

执行两条或两条以上的指令。

IPC——Instructions Per Cycle 的缩写。每个时钟周期完成的指令条数。

循环级并行性——循环的不同迭代之间存在的并行性。

指令的动态调度——是指在保持数据流和异常行为的情况下，通过硬件对指令执行顺序进行重新安排，减少数据相关导致的停顿。

指令的静态调度——指依靠编译器对代码进行静态调度，以减少相关和冲突。它不是在程序执行的过程中，而是在编译期间进行代码调度和优化的。

不精确异常——指当执行指令 i 导致发生异常时，处理机的现场（状态）与严格按程序顺序执行时指令 i 的现场不同。不精确异常使得在异常处理后难以接着继续执行程序。

精确异常——指当执行指令 i 导致发生异常时，处理机的现场跟严格按程序顺序执行时指令 i 的现场相同。

CDB——公共数据总线。

BHT——分支历史表。用来记录相关分支指令最近一次或几次的执行情况是成功还是失败，并据此进行预测。

分支目标缓冲——是一种动态分支预测技术。将执行过的成功分支指令的地址以及预测的分支目标地址记录在一张硬件表中。在每次取指令的同时，用该指令的地址与表中所有项目的相应字段进行比较，以便尽早知道分支是否成功，尽早知道分支目标地址，达到减少分支开销的目的。

ROB——ReOrder Buffer。前瞻执行缓冲器。

超标量——一种多指令流出技术。它在每个时钟周期流出多条的指令，但指令的条数不固定，依代码的具体情况而定，但有个上限。

超流水——在一个时钟周期内分时流出多条指令。

超长指令字——一种多指令流出技术。VLIW 处理机在每个时钟周期流出的指令条数是固定的，这些指令构成一条长指令或者一个指令包，在这个指令包中，指令之间的并行性是通过指令显式地表示出来的。

2. 填空题

【题 5.2】 答：动态开发、静态开发

【题 5.3】 答：基本程序块

【题 5.4】 答：记分牌方法、Tomasulo 算法

【题 5.5】 答：写结果、指令确认

【题 5.6】 答：乱序、程序顺序

【题 5.7】 答：保留站的编号、ROB

【题 5.8】 答：编译器、数据相关

【题 5.9】 答：转移、历史、方向

【题 5.10】 答：超标量、超长指令字 VLIW

3. 问答题

【题 5.11】 答：开发 ILP 的途径有两种，一种是资源重复，重复设置多个处理部件，让

它们同时执行相邻或相近的多条指令；另一种是采用流水线技术，使指令重叠并行执行。

【题5.12】 答：最关键的两个属性是：数据流和异常行为。

保持异常行为是指：无论怎么改变指令的执行顺序，都不能改变程序中异常的发生情况。即原来程序中是怎么发生的，改变执行顺序后是怎么发生的。这个条件经常被弱化为：指令执行顺序的改变不能导致程序中发生新的异常。

数据流是指数据值从其产生者指令到其消费者指令的实际流动。

【题5.13】 答：优点：①能够处理一些编译时情况不明的相关，并能简化编译器；②能够使本来是面向某一流水线优化编译的代码在其他动态调度的流水线上也能高效地执行。

但动态调度的这些优点是以硬件复杂性的显著增加为代价的。

【题5.14】 答：

(1) 指令状态表：记录正在执行的各条指令已经进入到了哪一段。

(2) 功能部件状态表：记录各个功能部件的状态。每个功能部件有一项，每一项由9个字段组成。

(3) 结果寄存器状态表Result：每个寄存器在该表中有一项，用于指出哪个功能部件(编号)将把结果写入该寄存器。

【题5.15】 答：其核心思想是：①记录和检测指令相关，操作数一旦就绪就立即执行，把发生RAW冲突的可能性减小到最少；②通过寄存器换名来消除WAR冲突和WAW冲突。寄存器换名是通过保留站来实现，它保存等待流出和正在流出指令所需要的操作数。

基本思想：只要操作数有效，就将其取到保留站，避免指令流出时才到寄存器中取数据，这就使得即将执行的指令从相应的保留站中取得操作数，而不是从寄存器中。指令的执行结果也是直接送到等待数据的其他保留站中去。因而，对于连续的寄存器写，只有最后一个才真正更新寄存器中的内容。一条指令流出时，存放操作数的寄存器名被换成为对应于该寄存器保留站的名称(编号)。

【题5.16】 答：从指令队列的头部取一条指令。如果该指令的操作所要求的保留站有空闲的，就把该指令送到该保留站(设为r)。并且，如果其操作数在寄存器中已经就绪，就将这些操作数送入保留站r。如果其操作数还没有就绪，就把将产生该操作数的保留站的标识送入保留站r。另外，还要完成对目的寄存器的预约工作，将其设置为接受保留站r的结果。

【题5.17】 答：

(1) 冲突检测逻辑和指令执行控制是分布的(通过保留站和CDB实现)。

每个功能部件的保留站中的信息决定了什么时候指令可以在该功能部件开始执行。如果有多条指令已经获得了一个操作数，并同时在等待同一运算结果，那么这个结果一产生，就可以通过CDB同时播送给所有这些指令，使它们可以同时执行。

(2) 消除了WAW冲突和WAR冲突导致的停顿。

这是通过使用保留站进行寄存器换名，并且在操作数一旦就绪就将其放入保留站来实现的。

【题5.18】 答：目的有两个：预测分支是否成功和尽快找到分支目标地址(或指令)，从而避免控制相关造成流水线停顿。

需要解决两个关键问题：①如何记录分支的历史信息；②如何根据这些信息来预测分支的去向，甚至提前取出分支目标处的指令。

【题5.19】 答：流水线3个阶段所进行的相关操作如图5.1所示。

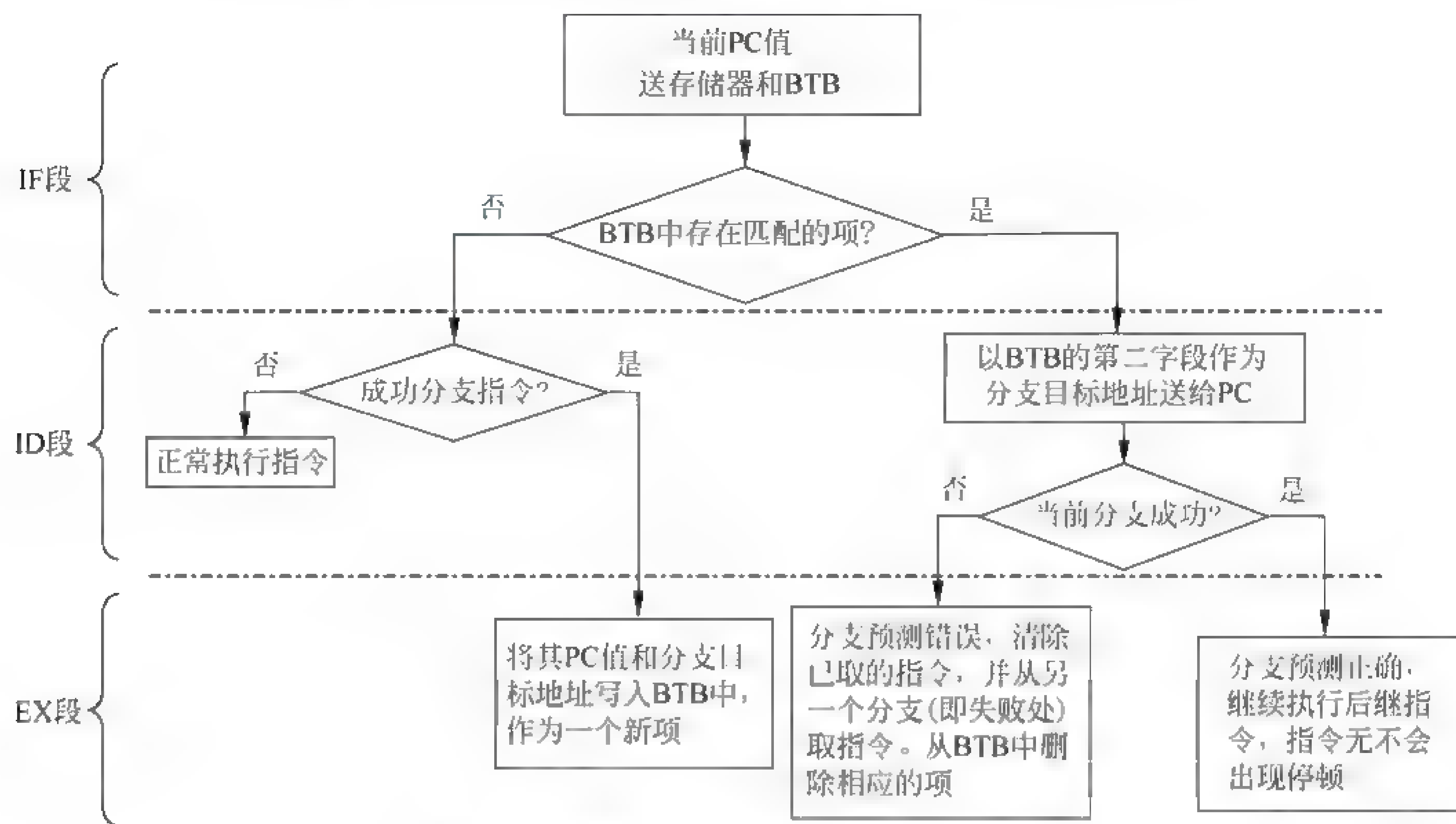


图5.1　流水线3个阶段所进行的相关操作

【题5.20】 答：

(1) 动态分支预测。用来选择后续执行的指令。

(2) 在控制相关的结果尚未出来之前，前瞻地执行后续指令。

(3) 用动态调度对基本块的各种组合进行跨基本块的调度。

【题5.21】 答：

(1) 指令类型：指出该指令是分支指令、store指令或寄存器操作指令。

(2) 目的地址：给出指令执行结果应写入的目的寄存器号(如果是load和ALU指令)或存储器单元的地址(如果是store指令)。

(3) 数据值字段：用来保存指令前瞻执行的结果，直到指令得到确认。

(4) 就绪字段：指出指令是否已经完成执行并且数据已就绪。

【题5.22】 答：这一阶段对分支指令、store指令以及其他指令的处理不同。

(1) 对于除分支指令和store指令以外的指令来说，当该指令到达ROB队列的头部而且其结果已经就绪时，就把该结果写入该指令的目的寄存器，并从ROB中删除该指令。

(2) 对store指令的处理与(1)类似，只是它是把结果写入存储器。

(3) 当预测错误的分支指令到达ROB队列的头部时，就表示是错误的前瞻执行。这时要清空ROB，并从分支指令的另一个分支重新开始执行。

(4) 当预测正确的分支指令到达ROB队列的头部时，该指令执行完毕。

【题5.23】 答：

(1) 超标量结构对程序员是透明的，处理机能自己检测下一条指令是否能流出，不需要

由编译器或专门的变换程序对程序中的指令进行重新排列。

(2) 即使是没有经过编译器针对超标量结构进行调度优化的代码或是旧的编译器生成的代码也可以运行，当然运行的效果不会很好。

【题5.24】 答：

(1) 程序所固有的指令级并行性。

(2) 硬件实现上的困难。多流出的处理器需要大量的硬件资源，随着每个时钟周期流出指令数的增加，所需的硬件成正比例地增长，所需的存储器带宽和寄存器带宽也大大增加了，这样的带宽要求必然导致大量增加硅片面积，加大面积就导致时钟频率下降、功耗增加、可靠性降低等一系列问题。

(3) 超标量和超长指令字处理器固有的技术限制。

4. 应用题

【题5.25】

解：

```
I1  LOAD  R1,M(A)
I2  LOAD  R2,M(B)
I3  LOAD  R3,M(C)
I4  LOAD  R4,M(D)
I5  MUL   R5,R1,R2
I6  ADD   R2,R3,R4
I7  ADD   R2,R2,R5
```

【题5.26】

解：(1) 指令 I_1 和 I_2 之间有 RAW 相关，I_2 和 I_3 之间有 RAW 相关，I_1 和 I_3 之间有 WAW 相关，I_1 和 I_2 之间还有 WAR 相关。

(2) 对 I_1 和 I_2 之间的 WAR 相关，可用定向传送解决。根据寄存器重命名技术，对引起 RAW 相关的 I_2 中的 R_2，对引起 WAW 相关的 I_3 中的 R_1，可分别换成备用寄存器 R_2'、R_1'。经寄存器重命名后，程序代码段实际执行时变为：

```
I1      ADD  R1,R2,R4
I2      ADD  R2',R1,1
I3      SUB  R1',R4,R5
```

【题5.27】

解：(1)程序执行的 CPI ＝没有分支的基本 CPI(1) ＋分支带来的额外开销

分支带来的额外开销是指在分支指令中，缓冲命中但预测错误带来的开销与缓冲没有命中带来的开销之和。

分支带来的额外开销＝15%×(90%命中×10%预测错误×4＋10%没命中×3)＝0.099

所以，程序执行的 CPI＝1＋0.099＝1.099。

(2) 采用固定的两个时钟周期延迟的分支处理 CPI＝1＋15%×2＝1.3

由(1)(2)可知分支目标缓冲方法执行速度快。

【题5.28】

解：设每条无条件转移指令的延迟为 x，则有：

$$1 + 5\% \times x = 1.1$$
$$x = 2$$

当分支目标缓冲命中时，无条件转移指令的延迟为 0。

所以，程序的 CPI＝1＋2×5％×(1－90％)＝1.01

【题 5.29】

解：(1) 表 5.3～表 5.5 给出了当采用 Tomasulo 算法时，在上述给定的时刻，保留站、load 缓冲器以及寄存器状态表中的内容。标志 Add1 表示是第一个加法功能部件，Mult1 表示是第一个乘法功能部件，其余以此类推。

表 5.3　指令执行状态

指　　令		指令执行状态		
		流出	执行	写结果
L. D	F6，34(R2)	√	√	√
L. D	F2，45(R3)	√	√	
MUL. D	F0，F2，F4	√		
SUB. D	F8，F2，F6	√		
DIV. D	F10，F0，F6	√		
ADD. D	F6，F8，F2	√		

表 5.4　保留站的内容

名称	保留站内容						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	yes	L. D					45+Regs[R3]
Add1	yes	SUB. D		Mem[34+Regs[R2]]	Load2		
Add2	yes	ADD. D			Add1	Load2	
Add3	no						
Mult1	yes	MUL. D		Regs[F4]	Load2		
Mult2	yes	DIV. D		Mem[34+Regs[R2]]	Mult1		

表 5.5　寄存器状态表中的内容

域	寄存器状态							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1	Load2		Add2	Add1	Mult2		

(2) MUL. D 指令准备写结果时各状态表的内容如表 5.6～表 5.8 所示。

这里，由于 ADD. D 指令与 DIV. D 指令的 WAR 冲突已经消除，ADD. D 可以先于 DIV. D 完成并将结果写入 F6，不会出现错误。

表 5.6　指令状态表的内容

指　令		指令状态表		
		流出	执行	写结果
L.D	F6, 34(R2)	√	√	√
L.D	F2, 45(R3)	√	√	√
MUL.D	F0, F2, F4	√	√	
SUB.D	F8, F6, F2	√	√	√
DIV.D	F10, F0, F6	√		
ADD.D	F6, F8, F2	√	√	√

表 5.7　保留站的内容

名称	保留站						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	no						
Add3	no						
Mult1	yes	MUL.D	Mem[45+Regs[R3]]	Regs[F4]			
Mult2	yes	DIV.D		MEM[34+Regs[R2]]	Mult1		

表 5.8　寄存器状态表

域	寄存器状态表							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1					Mult2		

【题 5.30】

解：状态表的内容如表 5.9～表 5.11 所示。这时指令 SUB.D 尽管已经执行完毕，但需要等到 MUL.D 得到确认后才能确认。

表 5.9　保留站的内容

名称	保留站							
	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL.D	Mem[45+Regs[R3]]	Regs[F4]			#3	
Mult2	yes	DIV.D		Mem[34+Regs[R2]]	#3		#5	

表 5.10　ROB

项号	ROB				
	Busy	指令	状态	目的	Value
1	no	L.D　F6，34(R2)	确认	F6	Mem[34+Regs[R2]]
2	no	L.D　F2，45(R3)	确认	F2	Mem[45+Regs[R3]]
3	yes	MUL.D　F0，F2，F4	写结果	F0	#2×Regs[F4]
4	yes	SUB.D　F8，F6，F2	写结果	F8	#1－#2
5	yes	DIV.D　F10，F0，F6	执行	F10	
6	yes	ADD.D　F6，F8，F2	写结果	F6	#4＋#2

表 5.11　浮点寄存器状态

字　段	浮点寄存器状态							
	F0	F2	F4	F6	F8	F10	…	F30
ROB 项编号	3			6	4	5		
Busy	yes	no	no	yes	yes	yes	…	no

【题 5.31】

解：(1)

```
L. D      F2, 0(R1)       1
Stall
MUT.D     F4, F2, F0      2
L. D      F6, 0(R2)       3
Stall
Stall
ADD.D     F6, F4, F6      4
Stall
Stall
S. D      F6, 0[R2]       5
DADDIU    R1, R1, #8      6
DADDIU    R2, R2, #8      7
DSUBIU    R3, R1, #DONE   8
BNEZ      R3, FOO         9
```

所以，共有 14 个时钟周期，其中有 5 个空转周期。

(2) 循环顺序展开 4 次，不进行任何指令调度，则指令 1～5 及其间的 stall 都是必要的，只是指令 6～9 只需执行一次，因此，共有 10×4＋4＝44 个时钟周期，计算出 4 个 Y 值，所以计算一个 Y 值需要 11 个时钟周期，加速比为：14/11＝1.27。加速主要是来自减少控制开销，即减少对 R1、R2 的整数操作以及比较、分支指令而来的。

(3) 循环顺序展开 4 次，优化和调度指令，如下。

```
L. D      F2, 0(R1)
L. D      F8, 8(R1)
L.D       F14, 16(R1)
L. D      F20, 24(R1)
MUT.D     F4, F2, F0
```

```
MUT.D     F10, F8, F0
MUT.D     F16, F14, F0
MUT.D     F22, F20, F0
L. D      F6, 0(R2)
L. D      F12, 8(R2)
L. D      F18, 16(R2)
L. D      F24, 24(R2)
ADD.D     F6, F4, F6
ADD.D     F12, F10, F12
ADD.D     F18, F16, F18
ADD.D     F24, F22, F24
S. D      F6, 0[R2]
S. D      F12, 8[R2]
S. D      F18, 16[R2]
S. D      F24, 24[R2]
DADDIU    R1, R1, #32
DADDIU    R2, R2, #32
DSUBIU    R3, R1, #DONE
BNEZ      R3, FOO
```

共用了 24 个时钟周期，则计算一个 Y 值平均需要 24/4=6 个时钟周期。

加速比：14/6=2.33

(4) 状态表的内容如表 5.12～表 5.15 所示。

表 5.12　指令执行时钟

指　令	指令执行时钟			
	流出	执行	写结果	确认
L. D　F2, 0(R1)	1	2	3	4
MUL. D　F4, F2, F0	2	4	5	6
L. D　F6, 0(R2)	3	4	6	7
ADD. D　F6, F4, F6	4	8	9	10
S. D　F6, 0(R2)	5	11	12	13
DADDIU　R1, R1, #8	6	7	8	
DADDIU　R2, R2, #8	7	8	9	
DSUBIU　R3, R1, #DONE	8	9	10	
BNEZ　R3, FOO	9	10		
L. D　F2, 0(R1)	10	11	13	14
MUL. D　F4, F2, F0	11	13	14	15
L. D　F6, 0(R2)	12	13	15	16
ADD. D　F6, F4, F6	13	17	18	19
S. D　F6, 0(R2)	14	20	21	22
DADDIU　R1, R1, #8	15	16	17	
DADDIU　R2, R2, #8	16	17	18	
DSUBIU　R3, R1, #DONE	17	18	19	
BNEZ　R3, FOO	18			

表 5.13　保留站

名称	保留站							
	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	yes	ADD. D	Regs[F4]	Regs[F6]				
Add2	no							
Add3	no							
Mult1	yes							
Mult2	no							

表 5.14　ROB

项号	ROB				
	Busy	指令	状态	目的	Value
1	yes	ADD. D　F6，F4，F6	执行	F6	Regs[F4]+Regs[F6]
2	yes	S. D　F6，0(R2)	流出	Mem[0+Regs[R2]]	#2

表 5.15　浮点寄存器状态

字　段	浮点寄存器状态							
	F0	F2	F4	F6	F8	F10	…	F30
ROB 项编号				1				
Busy				yes			…	

(5) 时钟周期数如表 5.16 所示。

表 5.16　时钟周期数

整 数 指 令	浮 点 指 令	时钟周期数
L. D　F2，0(R1)		1
L. D　F8，8(R1)		2
L. D　F14，16(R1)	MUT. D　F4，F2，F0	3
L. D　F20，24(R1)	MUT. D　F10，F8，F0	4
L. D　F6，0(R2)	MUT. D　F16，F14，F0	5
L. D　F12，8(R2)	MUT. D　F22，F20，F0	6
L. D　F18，16(R2)	ADD. D　F6，F4，F6	7
L. D　F24，24(R2)	ADD. D　F12，F10，F12	8
DADDIU R1，R1，#32	ADD. D　F18，F16，F18	9
S. D　F6，0(R2)	ADD. D　F24，F22，F24	10
S. D　F12，8(R2)		11
S. D　F18,16(R2)		12
S. D　F24，24(R2)		13
DADDIU R2，R2，#32		14
DSUBIU R3，R1，#DONE		15
BNEZ　R3，FOO		16

计算一个 Y 值需要 16/4=4 个时钟周期,加速比=14/4=3.5。

(6) 循环展开和指令调度如表 5.17 所示。

表 5.17　循环展开和指令调度

访存 1	访存 2	浮点指令 1	浮点指令 2	整数指令	时钟周期
L. DF2, 0(R1)	L. DF8, 8(R1)				1
L. DF14, 16(R1)	L. DF20, 24(R1)				2
L. DF6, 0(R2)	L. DF12, 8(R2)	MUT. DF4, F2, F0	MUT. DF10, F8, F0		3
L. DF18, 16(R2)	L. DF24, 24(R2)	MUT. DF16, F14, F0	MUT. DF22, F20, F0		4
		ADD. DF6, F4, F6	ADD. DF12, F10, F12		5
		ADD. DF18, F16, F18	ADD. DF24, F22, F24	DADDIU R1, R1, #32	6
				DADDIU R2, R2, #32	7
				DSUBIUR3, R1, #DONE	8
				BNEZR3, FOO	9
S. DF6, -32(R2)	S. DF12, -24(R2)				10
S. DF18,-16(R2)	S. DF24, -8(R2)				11

计算一个 Y 值需要 11/4 个时钟周期,加速比=14/(11/4)=56/11。

【题 5.32】

解:(1) 指令调度情况如表 5.18 所示。

表 5.18　指令调度(1)

第 一 路	第 二 路
LW R4, (R5)	
LW R7, (R8)	
DADD R9, R4, R7	LD R10, (R11)
DMUL R12, R13, R14	
DSUB R2, R3, R1	SW R15, (R2)
DMUL R21, R4, R7	SW R23, (R22)
SW R21, (R24)	

(2) 指令调度情况如表 5.19 所示。

表 5.19　指令调度(2)

第　一　路	第　二　路
LW　R4,(R5)	LW　R7,(R8)
DADD　R9,R4,R7	LD　R10,(R11)
DMUL　R12,R13,R14	DSUB R2,R3,R1
SW　R15,(R2)	DMUL　R21,R4,R7
SW　R23,(R22)	
SW　R21,(R24)	

(3) 指令调度情况如表 5.20 所示。

表 5.20　指令调度(3)

第　一　路	第　二　路
LW　R4,(R5)	LW　R7,(R8)
DSUB　R2,R3,R1	LD　R10,(R11)
SW　R23,(R22)	DMUL　R12,R13,R14
DADD　R9,R4,R7	DMUL　R21,R4,R7
SW　R15,(R2)	
SW　R21,(R24)	

【题 5.33】

解：标量流水处理机的时空图如图 5.2 所示。

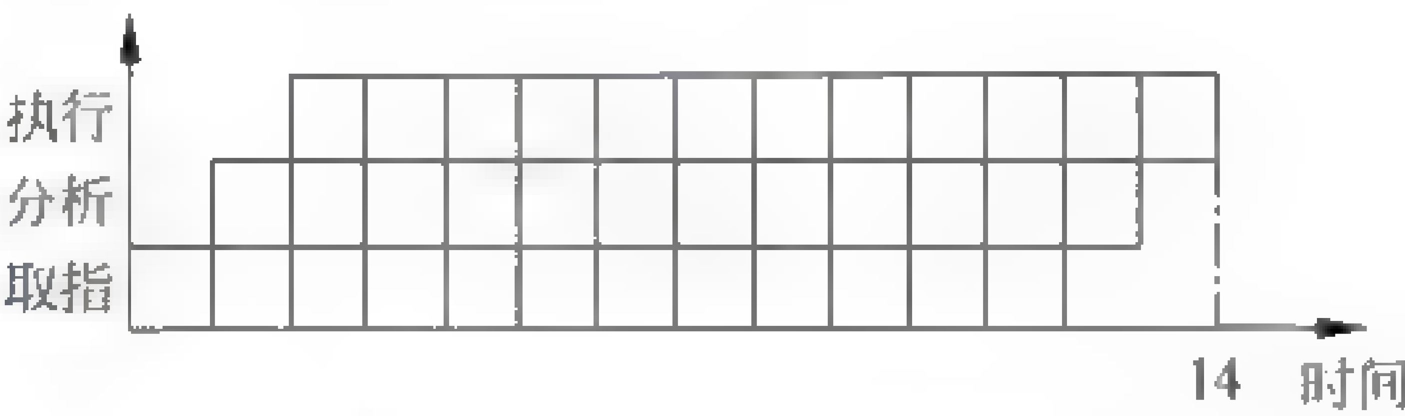


图 5.2　标量流水处理机

执行完 12 条指令需 $T_1=14\Delta t$。

超标量流水处理机与超长指令字处理机的时空图如图 5.3 所示。

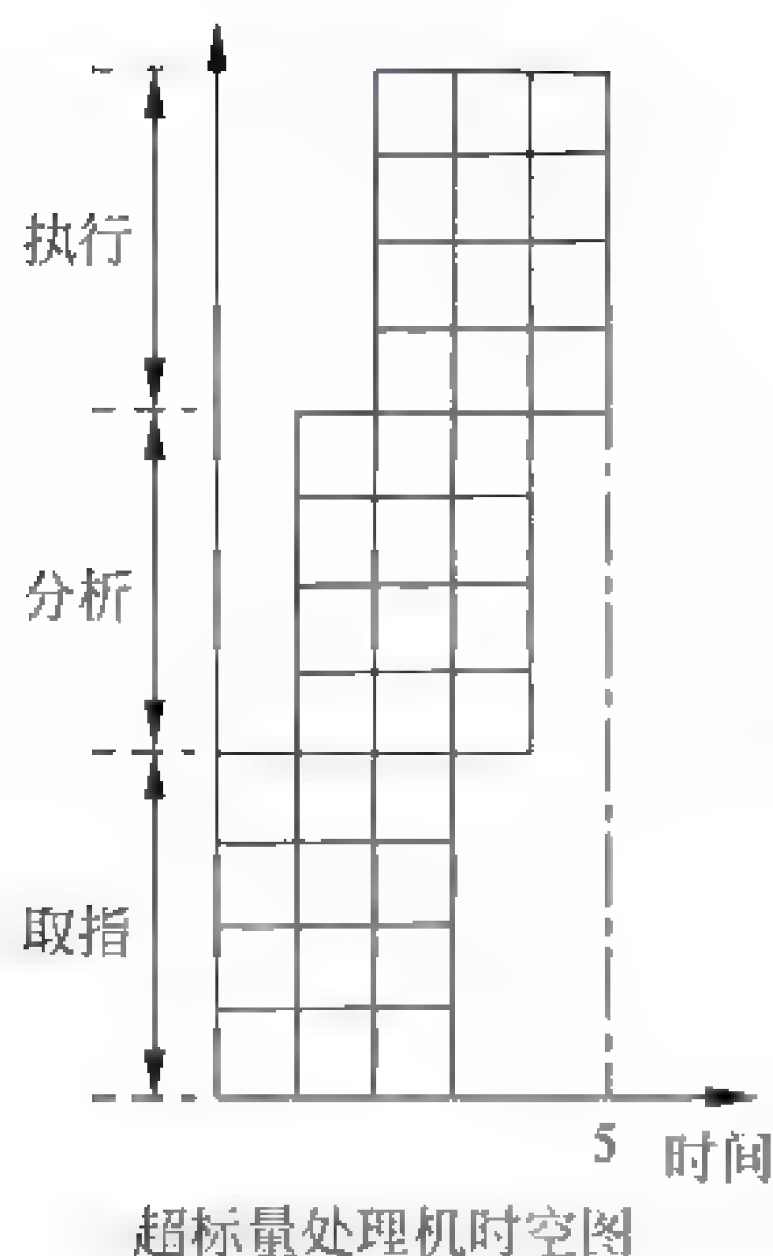
超标量流水处理机中，每一个时钟周期同时启动 4 条指令。执行完 12 条指令需 $T_2=5\Delta t$，相对于标量流水处理机的加速比为：

$$S_2=\frac{T_1}{T_2}=\frac{14\Delta t}{5\Delta t}=2.8$$

超长指令字处理机中，每 4 条指令组成一条长指令，共形成 3 条长指令。执行完 12 条指令需 $T_3=5\Delta t$，相对于标量流水处理机的加速比为：

$$S_3=\frac{T_1}{T_3}=\frac{14\Delta t}{5\Delta t}=2.8$$

超流水处理机的时空图如图 5.4 所示。



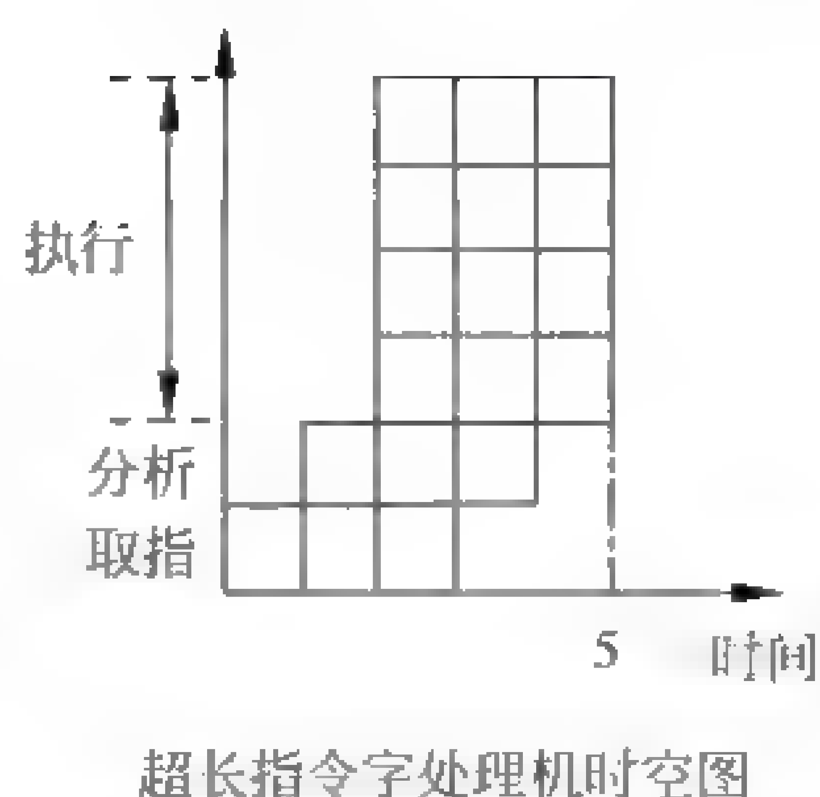


图 5.3　超标量流水处理机与超长指令字处理机的时空图

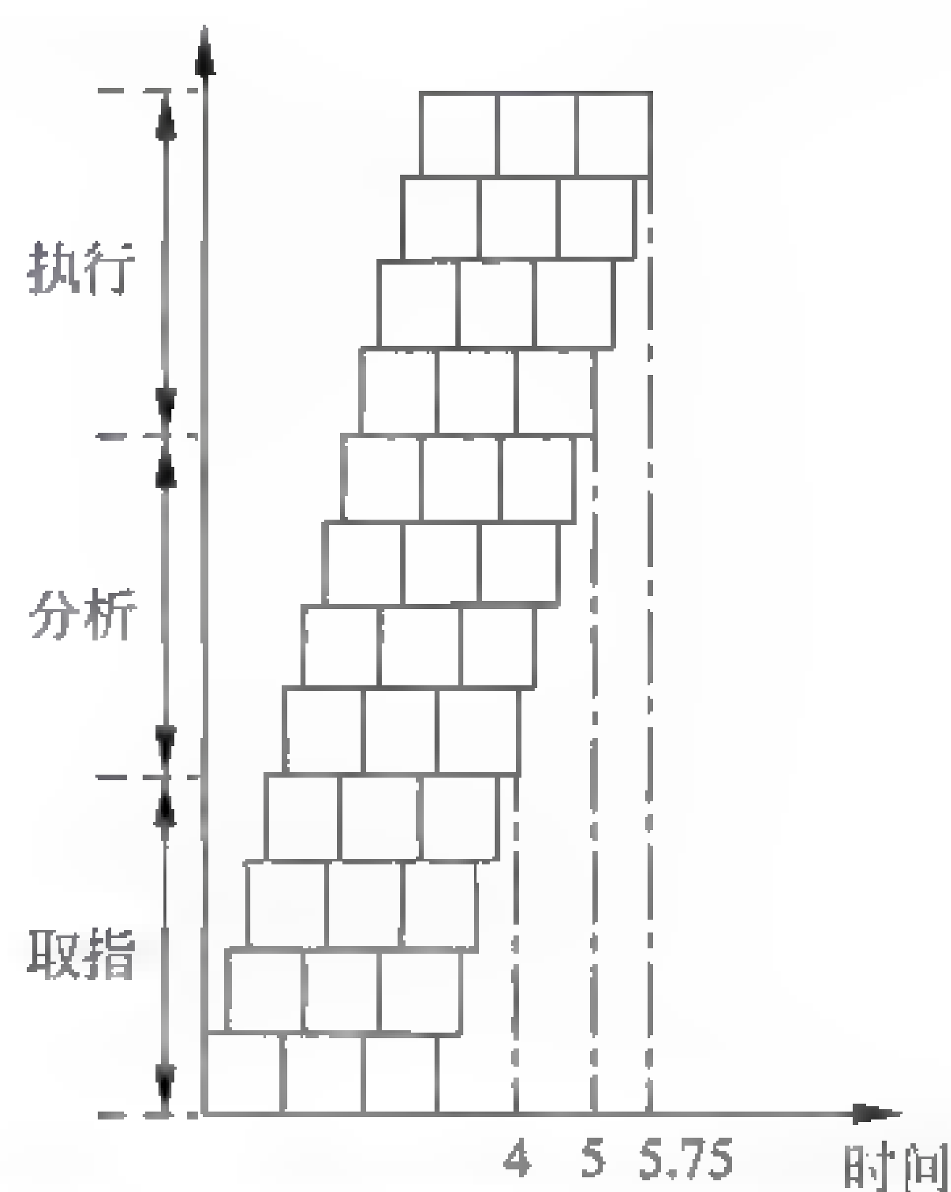


图 5.4　超流水处理机的时空图

超流水处理机中，每 1/4 个时钟周期启动一条指令。执行完 12 条指令需 $T_4=5.75\Delta t$，相对于标量流水处理机的加速比为：

$$S_4=\frac{T_1}{T_4}=\frac{14\Delta t}{5.75\Delta t}=2.435$$

【题 5.34】

解：(1) 指令 I_1、I_2 间有寄存器 R_0 的 WAR 相关；

指令 I_3、I_6 间有寄存器 R_2 的 WAR 相关；

指令 I_4、I_5 间有寄存器 R_4 的 RAW 相关；

指令 I_3、I_6 间有寄存器 R_2 的 WAW 相关。

(2) 采用顺序发射顺序完成调度方法的流水线时空图如图 5.5 所示。

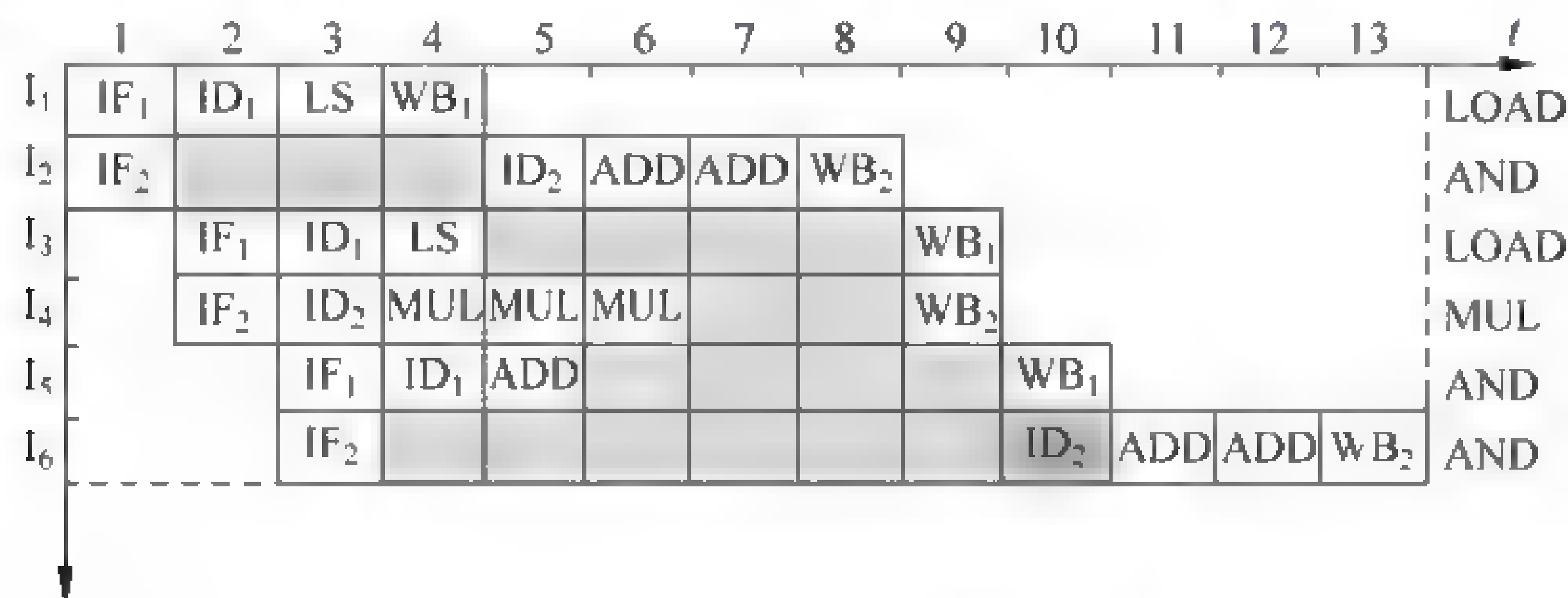


图 5.5　流水线时空图

从时空图看到，执行这个程序共用 130ns。

【题 5.35】

解：(1) 执行时，该循环将动态展开，并且只要可能就流出两条指令。从表 5.21 中可以看出，每 3 个时钟周期就执行一个新循环，每个循环 5 条指令，因此其 IPC 为：5/3=1.67。虽然指令的流出率比较高，但是执行效率并不是很高，16 拍共执行 15 条指令，平均指令执行速度为 15/16=0.94 条/拍。

表 5.21　基于 Tomasulo 算法的双流出超标量流水线中指令的流出、执行和写 CDB 的时间

遍数	指　　令		流出	执行	访存	写 CDB	说　　明
1	L. D	F0，0(R1)	1	2	3	4	流出第一条指令
1	ADD. D	F4，F0，F2	1	5		8	等待 L. D 的结果
1	S. D	F4,0(R1)	2	3	9		等待 ADD. D 的结果
1	DADDIU	R1，R1，#-8	2	4		5	等待 ALU(计算指令 S. D 的有效地址也是用该 ALU)
1	BNE	R1,R2，Loop	3	6			等待 DADDIU 的结果
2	L. D	F0，0(R1)	4	7	8	9	等待 BNE 完成
2	ADD. D	F4，F0，F2	4	10		13	等待 L. D 的结果
2	S. D	F4,0(R1)	5	8	14		等待 ADD. D 的结果
2	DADDIU	R1，R1，#-8	5	9		10	等待 ALU
2	BNE	R1,R2，Loop	6	11			等待 DADDIU 的结果
3	L. D	F0，0(R1)	7	12	13	14	等待 BNE 完成
3	ADD. D	F4，F0，F2	7	15		18	等待 L. D 的结果
3	S. D	F4,0(R1)	8	13	19		等待 ADD. D 的结果
3	DADDIU	R1，R1，#-8	8	14		15	等待 ALU
3	BNE	R1,R2，Loop	9	16			等待 DADDIU 的结果

(2) 资源使用情况如表 5.22 所示。

表 5.22　资源使用情况

时钟周期	整型 ALU	浮点 ALU	数据 Cache	CDB
2	1/L. D			
3	1/S. D		1/L. D	
4	1/DADDIU			1/L. D
5		1/ADD. D		1/DADDIU
6				
7	2/L. D			
8	2/S. D		2/L. D	1/ADD. D
9	2/DADDIU		1/S. D	2/L. D
10		2/ADD. D		2/DADDIU
11				
12	3/L. D			
13	3/S. D		3/L. D	2/ADD. D
14	3/DADDIU		2/S. D	3/L. D
15		3/ADD. D		3/DADDIU
16				
17				
18				3/ADD. D
19			3/S. D	
20				

【题 5.36】

解：功能部件状态表和结果寄存器状态表如表 5.23 和表 5.24 所示。

表 5.23　功能部件状态表

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L. D	F2	R3				no	
Mult1	yes	MULT. D	F0	F2	F4	Integer		no	yes
Mult2	no								
Add	yes	SUB. D	F8	F6	F2		Integer	yes	no
Divide	yes	DIV. D	F10	F0	F6	Mult1		no	yes

表 5.24　结果寄存器状态表

	结果寄存器状态表							
	F0	F2	F4	F6	F8	F10	…	F30
部件名称	Mult1	Integer			Add	Divide		

【题 5.37】

解：指令状态表、功能部件状态表和结果寄存器状态表如表 5.25～表 5.27 所示。

表 5.25　指令状态表

指　　令		指令状态表			
		流出	读操作数	执行	写结果
L. D	F6，34(R2)	√	√	√	√
L. D	F2，45(R3)	√	√	√	√
MULT. D	F0，F2，F4	√	√	√	
SUB. D	F8，F6，F2	√	√	√	√
DIV. D	F10，F0，F6	√			
ADD. D	F6，F8，F2	√	√	√	

表 5.26　功能部件状态表

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult1	yes	MULT. D	F0	F2	F4			no	no
Mult2	no								
Add	yes	ADD. D	F6	F8	F2			no	no
Divide	yes	DIV. D	F10	F0	F6	Mult1		no	yes

表 5.27　结果寄存器状态表

	结果寄存器状态表							
	F0	F2	F4	F6	F8	F10	…	F30
部件名称	Mult1			Add		Divide		

第 6 章　指令级并行的开发
——软件方法

6.1　基本要求与难点

6.1.1　基本要求

(1) 掌握相关的基本概念。

(2) 掌握循环展开法和指令调度的基本方法。

(3) 掌握全局指令调度技术：踪迹调度和超块调度。

(4) 理解静态多指令流出：VLIW 技术。

(5) 理解显式并行指令计算 EPIC 的概念，掌握非绑定分支、谓词执行和前瞻执行等技术。

(6) 理解能挖掘出更多的循环级并行的编译优化技术：循环携带相关，存储别名分析，数据相关分析。

(7) 掌握软流水技术。

(8) 了解 IA-64 体系结构。

6.1.2　难点

(1) 踪迹调度和超块调度。

(2) 显式并行指令计算 EPIC 的概念，非绑定分支、谓词执行和前瞻执行等技术。

(3) 软流水技术。

6.2　知识要点

我们将编译器所使用的指令级并行开发方法称作"软件方法"。与硬件方法相比，由于编译时能够"虚拟"出一个很大的指令窗口，软件方法有潜力开发出更多的指令级并行。

6.2.1　基本指令调度和循环展开

1. 指令调度的基本方法

为了充分发挥流水线的作用，必须设法让它满负荷地工作，这就要求充分开发指令之间

存在的并行性,找出不相关的指令序列,让它们在流水线上重叠并行执行,这一工作就是指令调度。本节讨论通过编译器来开发这种指令级并行的方法。

编译器完成指令调度的能力受限于两个特性:①程序固有的指令级并行;②流水线功能部件的执行延迟。在本节中,假设浮点流水线的延迟如表6.1所示。

表6.1 本节使用的浮点流水线的延迟

产生结果的指令	使用结果的指令	延迟(时钟周期数)
浮点计算	另一个浮点计算	3
浮点计算	浮点store(S.D)	2
浮点load(L.D)	浮点计算	1
浮点load(L.D)	浮点store(S.D)	0

编译时指令调度并不会真正消除指令间的相关,而是通过重新安排指令的流出顺序,使得指令间的相关尽可能少地引起流水线空转,从而减少整个指令序列在流水线上的执行时间。另一个需要注意的地方是,按照本节介绍的基本指令调度方法,指令调度不能跨越分支指令。

在许多情况下,每一个循环迭代中的指令条数不多,进行指令调度的余地很小。必须想办法增加每个循环迭代中的指令条数。循环展开(loop unrolling)就是解决这一问题的有效方法之一。

2. 循环展开

在前面讲过,增加指令间并行性最简单和最常用的方法,是开发**循环级并行**(loop level parallelism)——循环的不同迭代之间存在的并行性。所谓**循环展开**(loop unrolling)就是指把循环体的代码复制多次并按顺序排放,然后相应调整循环的结束条件。通过循环展开,多个循环迭代的代码可以合到一起调度,给编译器进行指令调度带来了更大的空间,而且还能够消除中间的分支指令和循环控制指令引起的开销。

通过循环展开、寄存器重命名和指令调度,可以有效地开发出指令级并行。

循环展开和指令调度时要注意以下几个方面。

(1) 保证正确性。在循环展开和调度过程中尤其要注意两个地方的正确性:循环控制,操作数偏移量的修改。

(2) 注意有效性。只有能够找到不同循环体之间的无关性,才能够有效地使用循环展开。

(3) 使用不同的寄存器。因为如果使用相同的寄存器,或者使用较少数量的寄存器,就可能导致新的冲突。

(4) 删除多余的测试指令和分支指令,并对循环结束代码和新的循环体代码进行相应的修正。

(5) 注意对存储器数据的相关性分析。例如,对于load指令和store指令,如果它们在不同的循环迭代中访问的存储器地址是不同的,它们就是相互独立的,可以相互对调。

(6) 注意新的相关性。由于原循环不同次的迭代在展开后都到了同一次循环体中,因此可能带来新的相关性。

6.2.2　跨越基本块的静态指令调度

一般来说，高效地优化含有分支结构的循环体需要在多个基本块间移动指令，这种调度被称为“全局指令调度”。本节介绍两种全局指令调度技术：踪迹调度和超块调度。

1. 全局指令调度

全局指令调度的目标是在保持原有数据相关和控制相关不变的前提下，尽可能地缩短包含分支结构的代码段的总执行时间。一般来说，对于单流出流水线，减少指令数就可以缩短总执行时间；而对于多流出处理，只有减少关键路径的长度才能真正缩短总执行时间。所谓**关键路径**(critical path)，是指根据指令间相关关系构成的数据流图中延迟最长的一条路径。只有缩短这条路径的长度，才能真正减少总执行时间。之所以要保持原有数据相关和控制相关不变，是为了保证执行结果正确。数据相关使得指令必须依照一定的先后顺序执行，而控制相关则限制了指令在基本块间的自由移动。

尽管前面介绍的循环展开技术可以减少构成循环的分支指令引起的流水线“空转”，但它却无法消除循环体内的分支指令引起的流水线“空转”。而全局指令调度技术则可以做到这一点，它的诀窍在于在循环体内的多个基本块间移动指令，从而扩大那些执行频率较高的基本块的体积。显然，这种技术的效果取决于能否准确估算各基本块的执行频率。不过，即便能够准确估算各基本块的执行频率，全局指令调度也无法保证总会带来性能提升，而且它还会带来新的问题，影响程序执行的正确性。

考虑以下代码：

```
          LD      R4,0(R1)        //取 A
          LD      R5,0(R2)        //取 B
          DADDU   R4,R4,R5        //A = A + B
          SD      R4,0(R1)        //存 A
          BEQZ    R4,thenpart     //A = 0 则转移
          X                       //代码段 X,基本块 elsepart
          J       join
thenpart:                         //基本块 thenpart
          SD      …,0(R2)         //指令 I1
join:
          SD      …,0(R3)         //指令 I2
```

有一些复杂的调度机制可以确保在调度 I_1 的情况下保持执行结果不变，但需要向基本块 elsepart 中增加若干指令以保证结果正确。这些新增加的指令被称为补偿代码(compensation code)。不过这种调度在编译实现时比较复杂，而且当 BEQZ 指令转移不成功时补偿代码的执行会带来额外的时间开销。

进行全局指令调度时编译器必须仔细斟酌哪些指令可被选作调度的对象。选择的依据很简单，即调度这条指令是否一定会带来性能提升。例如，将指令 I_1 调度到 BEQZ 之前必须判断这种调度是否一定能够缩短总执行时间。答案是不一定，因为为了保证执行结果的正确需要增加一些补偿代码，补偿代码的执行会带来额外的时间开销。同样，调度指令 I_2 时

也面临同样的问题。全局指令调度只能告诉我们将指令调度到分支指令之前带来性能提升的概率比较大。

在调度指令 I_1 之前,编译器需要综合考虑以下多个因素。

(1) 该分支中基本块 thenpart 和 elsepart 的执行频率各是多少?如果 thenpart 执行得更加频繁,那么调度 I_1 很可能带来性能提升;否则,尽管也可以调度 I_1,但调度后的代码执行效率反而会降低。

(2) 在分支语句前完成 I_1 所需的开销是多大?如果分支语句前有一些"空转"周期,而且 I_1 可以被调度到某个这样的周期内执行,那么所需的开销为0。

(3) 调度 I_1 是否能够缩短 thenpart 块的执行时间?如果 I_1 是该块关键路径的第一条语句,调度它可以减少 thenpart 块的执行开销。

(4) I_1 是否是最佳的被调度对象?调度 I_2 或 thenpart 块内的其他指令是否能够获得更大的性能提升?

(5) 若需要向 elsepart 块中增加补偿代码,补偿代码的执行开销是多少?怎样生成补偿代码?

可以看出,全局指令调度问题非常复杂,不仅选择哪条指令作为调度对象取决于多个因素,而且各条被调度的指令往往也是紧密相关的。即便是确定从哪条指令开始进行全局指令调度也是一个十分复杂的问题。

为了简化这一过程,人们提出了一些新的全局指令调度技术。接下来将详细讨论其中的两种:踪迹调度和超块调度,它们的共同之处在于将编译优化的重点放在那些执行频率很高的路径上,将这些路径上的基本块拼接在一起构成一个更大的基本块并进行优化。显然,如何处理分支指令是这些技术必须解决的一个关键问题。

2. 踪迹调度

踪迹调度(trace scheduling)非常适合多流出处理器,因为对于这些处理器而言,简单地进行循环展开和基本指令调度很难开发出足够的指令级并行来使流水线保持在充满状态。**踪迹**(trace)是程序执行的指令序列,通常由一个或多个基本块组成,trace 内可以有分支,但一定不能包含循环。踪迹调度会优化执行频率高的 trace,减少其执行开销。但由于需要添加补偿代码以确保正确性,那些执行频率较低的 trace 的开销反而会有所增加。可见,仅当不同 trace 的执行频率差别较大而且各条 trace 的执行频率受输入集的影响较小时,这种方法才能取得比较好的效果。这使得踪迹调度一般只适用于特定类型的应用。

踪迹调度过程分为两步。第一步称为**踪迹选择**(trace selection),负责从程序的控制流图中选择执行频率较高的路径,每条路径就是一条 trace。循环展开是生成 trace 的常用方法之一,它主要处理控制循环的分支指令,因为在一般情况下循环体的执行频率会远远高于循环体外其他基本块的执行频率。至于其他分支指令,一般是根据静态分支预测的结果或是在典型输入集下的执行统计信息进行处理,若转移成功(或失败)的概率很高,则将其视作转移总是成功(或失败)。显然,进行踪迹选择时,我们只能考虑转移成功和失败的概率相差很大的分支指令,而那些转移成功和失败概率接近的分支指令,可以通过谓词执行技术进行优化。

生成 trace 后就可以开始第二个步骤了,这一步称为**踪迹压缩**(trace compaction),即对

已生成的 trace 进行指令调度和优化，尽可能地缩短其执行时间。进行踪迹压缩时采用的方法与前面介绍的基本指令调度相似，都是通过重新安排 trace 内指令的执行顺序来缩短总执行时间。但是，当跨越 trace 内部的入口或出口调度指令时必须非常小心，因为这有可能改变原有的控制相关和数据相关，从而造成执行结果错误。通常需要向 trace 外该入口的前驱基本块或该出口的后继基本块中增加补偿代码，以确保执行结果的正确。

在踪迹调度中，由于选出的 trace 都是执行频率很高的路径，减少它们的执行开销有助于缩短程序的总执行时间，这就是踪迹调度能够提升性能的最根本原因。目前踪迹调度技术已经成功应用于对科学计算程序的优化中，这些应用中都包含大量的循环，而且能够准确地统计出程序运行的行为特征。

3. 超块调度

在踪迹调度中，如果 trace 入口或出口位于 trace 内部，编译器生成补偿代码的难度将大大增加，而且编译器很难评估这些补偿代码究竟会带来多少性能损失，这是踪迹调度的一个重要缺点。为解决这个问题，人们增加了对 trace 拓扑结构的约束，将其限制为只能拥有一个入口，但可以拥有多个出口的结构，这种新结构被称为**超块**(superblock)。超块可被视作一种扩展的基本块结构，显然其构造过程与 trace 非常相似。

由于只有一个入口，压缩超块比压缩 trace 容易得多，因为在压缩超块时，只需要考虑跨越超块出口移动指令的情形，而且对于那些只有一个出口的计数控制循环(例如只有一个循环 100 次后结束的 for 循环)，经过循环展开后得到的超块只有一个入口和一个出口，在这样的超块中进行指令调度显然更加容易。

与踪迹调度相比，超块调度可以简化补偿代码生成过程，并降低指令调度的复杂度，但由于其结构的限制(只有一个入口)，超块结构的目标代码体积也会大大增加。此外，它面临着与踪迹调度相同的问题——补偿代码的生成使得编译过程更加复杂，而且由于无法准确评估由补偿代码引起的时间开销，这种方法的应用范围受到一定限制。

踪迹调度、超块调度等全局指令调度技术的目的都是尽可能地开发更多的指令级并行，因而更加适用于多流出处理器。但多流出处理器如何充分利用这些编译时开发出的指令级并行呢？后面将详细讨论这一问题。

6.2.3　静态多指令流出：VLIW 技术

超标量处理器在运行时动态确定指令窗口中哪些指令可以被流出执行，为此它必须准确识别出指令窗口内的指令以及流水线上的所有指令之间存在着哪些相关。在动态调度的超标量处理器中，这些工作基本都是由硬件完成，而在静态调度的超标量处理器中，部分相关检测和指令调度工作交由编译器完成，大大降低了硬件实现的复杂度。

与超标量处理器不同，VLIW(超长指令字)处理器在编译时静态确定哪些指令能够同时流出，进一步降低了流水线硬件的实现复杂度，这有助于提高它的主频。VLIW 能够把同时流出的或者满足特定约束的一组操作打包在一起，得到一条更长(64 位、128 位或更长)的指令，这就是 VLIW 名字的由来。每个操作被放在 VLIW 指令的一个槽(slot)内。VLIW 处理器执行这样一条长指令就相当于超标量处理器同时执行多条指令，从而实现了多流出。

由于所有开发指令级并行的任务都交由编译器完成,VLIW处理器需要更加"智能"的编译器。

只有从应用程序中挖掘出足够多的并行指令打包到VLIW指令中,才能提高VLIW处理器中功能单元的利用率,充分发挥VLIW处理器的性能优势。使用6.1节介绍的基本指令调度和循环展开技术以及6.2节介绍的全局指令调度技术都能够有效识别哪些指令可以并行执行。

造成VLIW目标代码编码效率低的原因主要有两个:①为了消除流水线"空转"需要增加循环展开的次数,这增加了目标代码的体积;②很难从应用程序中找到足够多的并行指令填满VLIW指令中的每一个槽。使用代码压缩/还原技术可以减少VLIW目标代码体积过大带来的性能损失:VLIW目标代码被压缩后保存在硬盘中,只保留指令中非nop操作的信息;程序运行时,当代码被加载到指令Cache时或译码时再将其解压缩,还原出所含的nop操作。

为了简化硬件实现,很多VLIW处理器中没有实现任何相关检测逻辑,而是靠互锁机制保证执行结果的正确。当一个功能单元暂停时,互锁机制将暂停整个流水线,从而保证所有功能单元的同步。使用互锁机制的主要原因是一些操作的延迟在编译时无法确定,比如load操作,由于编译时无法确定它访问Cache是否命中,因而无法确定其延迟。出于性能上的考虑,编译时通常会假定这类操作的延迟为可能的最小值,如load操作的延迟是Cache的命中时间。这样,若运行时访问Cache不命中,互锁机制将暂停整个流水线,直至load操作完成。当VLIW指令中含有较多数量的操作槽且应用程序中的访存操作较多时,这种简单的互锁机制将造成较大的开销。现有的一些高性能VLIW处理器通常采用软硬件结合的方式解决这一问题:编译器负责保证同时流出的操作间没有相关,而流水线硬件则负责确保正在运行的指令在不满足上述同步约束的情况下也能得到正确的执行结果。

目标代码兼容性差是VLIW的另一个严重缺陷,极大地制约了VLIW的推广与应用。VLIW的指令格式与操作类型、功能单元的数量以及延迟等体系结构参数密切相关,当这些参数发生变化时,VLIW的指令格式也将相应地发生变化。要在同一系列不同代的处理器之间实现目标代码兼容,VLIW比超标量困难得多。二进制翻译或仿真是解决VLIW目标代码兼容问题的可行方法之一。**二进制翻译**(binary translation)是指将某个硬件平台的二进制目标代码翻译为另一个平台的目标代码的过程,是目前在不同平台之间实现目标代码兼容的主要手段之一。二进制翻译可以在编译时静态完成,也可以在运行时动态完成,是目前计算系统虚拟化研究的主要内容之一。

6.2.4 显式并行指令计算

超标量和VLIW是开发指令级并行的两种极端结构,前者完全依赖流水线硬件动态识别出可并行的指令,并将它们分发给相应的功能单元执行,后者则将指令级并行的开发工作全部交给编译器完成,在编译时静态确定每条指令的流出时刻和执行延迟,仅依赖简单的流水线硬件确保在指令实际执行延迟与编译器假定的延迟不一致时(如访问Cache不命中就会增加访存操作的延迟),程序的执行结果依然正确。在这两种结构中,单一的指令级并行开发机制使得它们都存在着严重的固有缺陷:超标量结构硬件复杂度太高,学术界和工业

界一致认为，同时流出并执行 8 条指令将达到这种结构的极限；VLIW 则面临着严重的代码兼容问题，而且目前 VLIW 编译器的智能程度远远无法满足人们的要求。**显式并行指令计算**(Explicitly Parallel Instruction Computing，EPIC)技术正是为了解决这两种结构的本质缺陷而提出的，它是在 VLIW 的基础上融合了超标量结构的一些优点而设计得到的，以期用有限的硬件开销为代价开发出更多的指令级并行。

EPIC 结构充分利用编译器和流水线硬件的协同能力开发更多的指令级并行。编译器根据对程序运行特征的统计信息，如分支指令转移成功的概率、访问 Cache 命中的概率等，通过踪迹调度、超块调度等带有极强猜测性的编译优化技术从应用程序中尽可能多地挖掘指令级并行，流水线硬件则提供丰富的计算资源实现这些指令级并行，并通过专门的机制确保在程序执行过程中出现预测错误时仍然能得到正确的运行结果，尽量减少由此引起的额外开销。

一般来说，EPIC 结构必须符合以下两个基本特点。第一，指令级并行主要由编译器负责开发，处理器应为保证代码正确执行提供必要的硬件支持，只有在这些硬件机制的辅助下这些优化技术才能高效完成。第二，系统结构必须提供某种通信机制，使得流水线硬件能够了解编译器“安排”好的指令执行顺序。但需要注意的是，EPIC 并不仅仅是采用了多种高级编译优化技术的 VLIW 结构，这只不过是它的一个特征。EPIC 的第二个特征，有效的软硬件通信机制，才是它与 VLIW 之间的本质区别。

EPIC 是在 VLIW 的基础上设计得到的，因此它的指令格式也与 VLIW 相似，编码效率也比较低。特别是为了填充分支延迟槽，需要大量的分支无关操作，进一步降低了代码效率，而且随着处理器主频的提高，分支延迟(以时钟周期为单位)越来越大，这一问题变得更加严重。为解决这一问题，EPIC 编译器采用了多种高级优化技术，如非绑定分支、谓词执行、前瞻执行等，处理器则提供必要的硬件支持，以保证经过这些优化后的代码能够正确执行。

1. 非绑定分支

分支指令在执行时必须完成一系列“动作”：计算分支转移条件、生成分支目标地址、取下一条指令、译码并流出下一条指令。尽管在传统指令系统中，每一条分支指令都被视作“原子的”而独立存在，但它完全可以被划分为多条粒度更小的指令。

非绑定(unbundled)分支技术就是基于这一思想而提出的，它将一条分支指令划分为 3 个独立的操作并进行调度。这 3 个操作是：①准备操作，计算分支目标地址；②比较操作，计算分支转移条件；③转移操作，根据分支转移条件是 true 还是 false，或改变控制流或执行顺序的下一条指令。运行时，流水线硬件根据前两个操作的结果，动态地将第三个操作转换为空操作或无条件转移。显然，前两个操作完成得越早，流水线硬件改变控制流的时间就越充裕，分支操作引起流水线“空转”的可能性就越小。例如，在得到转移成功目标地址后，指令预取模块就可以提前从该地址处预取指令，而在计算出分支转移条件后，就可以确定之前进行的预取是否正确，并进行相应的处理。分支转移条件为 false 就可以作废这次不必要的预取。

2. 谓词执行

当分支指令转移成功的概率远大于或远小于转移失败的概率时，在编译时静态预测这

些分支指令的行为相对容易一些，此时可以通过循环展开和全局指令调度技术扩大基本块的体积，以便开发更多的指令级并行。若分支指令转移成功与失败的概率比较接近，或是分支指令的行为受输入集影响很大，这些编译优化技术的效果就没有那么好了，控制相关将成为限制指令级并行开发的主要原因，谓词执行技术能够很好地解决这一问题。

谓词执行是一种特殊的条件执行机制。所谓条件执行，是指指令的执行依赖于一定的条件，当条件为真时指令将正常执行，否则将什么也不做。MIPS、PowerPC、SPARC、x86 等很多处理器都实现了条件传输指令，这条指令就是按条件执行的，只有在执行条件为真时才会进行数据传输。分支转移条件是最常见的一种指令执行条件，在“if-then-else”分支结构中，若分支转移条件为真，将执行 then 部分的指令，else 部分的指令根本不会被读出。但在条件执行机制下，无论指令的执行条件是否为真，指令都将被读出、译码并执行，这是二者之间最重要的区别。

借助条件传输指令，可以将分支指令引起的控制相关转换为相对于分支转移条件(R1)的数据相关。对于流水线处理器而言，这种转换的意义在于，那些本应在流水线前端 ID 段处理的控制相关被推迟到流水线后端 WB 段处理，为指令调度提供了更大的空间。而且，借助这种转换，可以删除代码中那些行为难以预测的分支指令，提高分支预测准确率，并减少由于分支预测错误带来的性能损失。这种转换技术称作 **if 转换**(if conversion)。

利用条件传输指令可以简化求绝对值的运算 A＝abs(B)，它对应的高级语言语句为：

```
if(B<0){A=-B;} else {A=B;}
```

这个 if then else 结构可以被等价地转换为下面的代码段，假设变量 A、B 分别被保存在寄存器 R1 和 R2 中：

```
SUB      R1,R0,R2      //A=-B
SLT      R3,R2,R0      //(B<0)?,若 B<0,R3=1,否则 R3=0
CMOVZ    R1,R2,R3      //R3=0 时,A=B
```

在这段代码中，第一条指令给寄存器 R1 赋初值(-B)，第二条指令负责计算传输条件并将其保存在寄存器 R3 中，二者可以并行。第三条指令则根据传输条件修改 R1 的值，当 R3＝0 即 B≥0 时，R1 将被修改为 B，这样就完成了求绝对值的操作。

随着流水线中同时流出的指令数不断增加，处理器设计者必须做出选择：究竟是增加分支处理单元，在一个周期内完成多条分支指令，还是采取类似条件执行这样的机制来消除代码中的分支指令。同时执行多条分支指令难度很大，这不仅是因为这些分支指令之间也存在着控制相关，而且在于流水线硬件在实现同时预测两个分支的行为、同时更新分支预测表两项操作时也具有相当的难度。因此，很多设计者放弃了第一种想法，转而利用条件执行机制减少分支指令引起的流水线性能损失。

条件传输是最简单的条件执行机制，当一个分支结构的 then 部分和 else 部分中仅含有少量指令时，这种机制的效果较好，就像前面的例子那样。但随着指令数的增加，经过 if 转换得到的条件传输指令和条件计算指令的数量也将增加，这会大大降低目标代码的效率。一些处理器采用谓词执行方式来解决这一问题。

谓词执行(predicated execution)是给指令集中的每条指令都增加一个执行条件，这个执行条件就叫作**谓词**(predicate)。若谓词为真，指令将正常执行，否则将什么也不做，就像

条件传输指令那样。在谓词执行机制的支持下，一个 if-then-else 分支结构的 then 部分与 else 部分中的每条指令都可以被转换为谓词指令，then 部分中所有指令的谓词就是分支转移条件，而 else 部分中指令的谓词正好与之相反。谓词执行机制能够显著增加全局指令调度的效果，因为从理论上来说，if 转换可以删除所有不构成循环的分支指令，而从前面各节的讨论中我们已经知道，它们才是全局指令调度面临的最大障碍。

谓词执行增加了异常处理机制的复杂度。不过，谓词执行机制最复杂的地方还不是异常处理，而是决定何时将执行条件不成立的谓词指令转换为空操作。这可以在两个不同时机进行：流水线前端指令流出时，或是流水线后端结果确认时。但遗憾的是，每种方式都有其不足之处。若要在指令流出前决定是否将其转换为空操作，必须尽早知道谓词的值，也就是分支转移条件的值，可这个值是很难预测的。这样，在谓词指令与计算分支条件的那条指令之间就存在数据相关，这个数据相关极有可能引起流水线"空转"。第二种方式的问题在于它降低了功能单元的实际利用率——由于结果没有被写回，功能单元做了无用功。不过，对于多流出处理器而言这对性能的影响并不大，因为在很多情况下即使不执行这些谓词指令，功能单元也将处于空闲状态。这也就是为什么现有的支持谓词执行机制的处理器都选择第二种实现方式的主要原因。

3. 前瞻执行

借助谓词执行机制可以实现优化。不过目前仅有 IA 64、ARM 等少量系统结构全面实现了谓词执行（或条件执行）机制，绝大多数系统结构只是部分地进行了实现，如 MIPS、PowerPC、SPARC、Intel x86 等结构只是简单地实现了条件传输指令。在这些平台上，尽管可以借助全局指令调度技术跨越条件分支进行指令调度，将部分指令调度到分支指令之前猜测执行，但为了确保结果的正确性，必须增加补偿代码，这降低了猜测执行的性能受益。EPIC 结构则通过前瞻执行机制解决这一问题。

前瞻（speculation）执行并不仅仅是在数据相关或控制相关尚未消除的情况下，简单地将指令调度到与之相关的指令之前猜测执行，它还通过一系列复杂的硬件机制完成异常处理、正确性保证等工作。一般来说，影响前瞻执行效果的因素主要有以下 3 个。

（1）编译器能力的大小，即编译器能否准确识别出可以前瞻执行的指令，并在保持程序数据流不变的前提下将它们移到分支指令之前。有时可能需要进行寄存器重命名。

（2）异常处理机制能否推迟处理由被前瞻执行的指令引起的异常，直到确定前瞻指令确实应该被执行后。在谓词执行机制中也存在类似的问题。

（3）如何避免前瞻引起的错误。举个例子，为了减少访存指令引起的流水线"空转"，有时编译器会将 load 指令调度到相邻的上一条或几条 store 指令之前执行，若发现前后两个访存指令的地址相同，如何保证程序执行结果依然正确？

第一个因素取决于究竟采用了何种编译优化技术，在本章前面的讨论中已经分析过这一问题。本节将着重针对后两点进行深入讨论。

当前瞻执行的指令发生异常时，该怎么处理呢？一般说来，有 4 种方法可以解决这一问题。在详细讨论这 4 种方法之前，我们先将程序产生的异常分为两类，第一类异常发生时程序执行将被迫终止，如违反存储保护权限（memory protection violation）异常，它们被称为"终止性（terminal）异常"；另一类异常叫作"可继续（resumable）异常"，在被处理后程序仍

可继续执行，如缺页。显然，必须仔细处理由错误前瞻的指令引发的第一类异常。

最简单的方法是立即处理每个异常，不管它们是否由被前瞻执行的指令引发。为避免前瞻指令引起的终止性异常造成程序执行结束，对于此类异常只是简单地返回一个未定义值即可，而不是立即结束程序的运行。这样，当前瞻正确时，正在执行的程序不会被终止，但它的执行结果肯定是错误的；而当前瞻错误时，程序也将继续执行下去，只是处理该异常的返回值不会被使用。按照这种处理方式，若程序的逻辑功能正确，它的运行结果也将是正确的，否则程序虽然不会在运行时终止，也只能得到错误的运行结果。对于某些应用程序来说，这种处理方式是不可取的，但也有一些处理器根据这一思想提供了快速异常处理模式。

第二种方法则借助专门的检测指令判断是否需要进行异常处理，下面的 MIPS 汇编代码就使用了这样一条指令 SPECCK：

```
        LD      R1,0(R3)        //取 A
        sLD     R14,0(R2)       //取 B,前瞻执行
        BEQZ    R1,L1
        SPECCK  0(R2)           //检查指令 sLD 是否产生异常
        J       L2
L1:     DADDI   R14,R1,#4       //A = A + 4
L2:     SD      R14,0(R3)       //A = B
```

分支转移失败意味着指令 sLD 应该被执行，SPECCK 指令将检查 sLD 执行时是否已引发异常，并进行相应的处理。这种方法的关键在于，推迟处理由前瞻指令引发的异常，直到已确定该指令确实应该被执行。

引入 SPECCK 指令后会影响目标代码的效率，为此人们又提出了第三种处理方法，它的工作原理也十分简单。首先为每个通用寄存器增加一个特殊的状态标志位——“poison”位。前瞻指令引发的可继续异常都将被立即处理，而当它引发终止性异常时，其目的寄存器 R 的 poison 位将被置 1，否则该位将被清 0。接下来，当前瞻指令之后的另一条指令访问 R 时，若 R 的 poison 位为 1 将触发一个终止性异常。可见，这种方法利用寄存器的 poison 位记录了以该寄存器为目标寄存器的前瞻指令在执行时是否引起异常，并将异常处理推迟到另一条指令访问该寄存器时。

最后一种方法与第 5 章介绍的基于硬件的前瞻(hardware speculation)机制相似，也是将指令的执行结果保存在再定序缓冲器内，并按指令流出的顺序依次确认，但是前瞻指令的确认时机却被推迟，直至能够确定该指令的前瞻执行是正确(或错误)的。例如，一条指令 I 被前瞻调度到分支指令 B 之前执行，当 B 的结果被确认后，就可以知道 I 的这次前瞻执行是成功还是失败了。如果 I 本应被执行(前瞻成功)，而且在前瞻执行时触发了终止性异常，程序将被立即终止；如果 I 本不应被执行(前瞻失败)，I 引发的所有异常都将被取消。这种处理机制除了需要再定序缓冲器等硬件机制的支持外，也需要在编译时标出所有被前瞻的指令，以及这些指令所跨越的条件分支。尽管全局指令调度技术允许跨越多个条件分支进行指令调度，但测试结果却表明，跨越一个条件分支进行前瞻调度即可获得与跨越多个分支调度接近的性能加速比，这样编译器只需 1 个比特(bit)即可指明被前瞻指令的执行条件，该位为 1 表示该指令来自分支结构的 then 部分，为 0 则表示该指令来自分支结构的 else 部分。

到目前为止，本节讨论的前瞻执行都属于控制前瞻，即将指令调度到条件分支之前执

行，这种前瞻改变了原有的控制相关。下面讨论数据前瞻的处理，这是由于指令调度破坏了指令间的数据相关而引起的。

将 load 指令调度到 store 指令之前执行是最常见的数据前瞻调度。之所以这样进行前瞻调度，主要原因有两个：①由于缺乏足够的运行时信息，如不知道寄存器的值，编译时很难确定两次访存是否会访问相同的存储单元。为保证正确性，编译器总是会选择保守的调度方法，认为二者存在地址冲突，但在很多情况下，地址并不冲突。②尽早完成 load 指令有助于缩短关键路径的长度。

由于编译器仅负责将 load 指令调度到 store 指令之前，检测地址冲突的任务就交给流水线硬件完成。检测方法为：当 load 指令前瞻执行时，流水线硬件会将它访存的地址记录在一个特殊的地址表中；接下来，每执行一条 store 指令，流水线硬件将该指令的访存地址与地址表中的各有效项进行匹配，命中则说明出现地址冲突，前瞻失败；若控制流抵达 load 指令原来所在的位置时未出现冲突，说明前瞻成功，流水线硬件从地址表中删除对应的项。编译器负责把一条特殊的检测指令放在 load 指令原来所在的位置，执行到这条指令时若没有出现地址冲突，则说明 load 指令前瞻成功。

前瞻失败的处理方式有两种。如果仅有 load 指令被前瞻执行，这是最简单的情况，只需由检测指令重新执行这次 load 操作即可。但如果数据相关于该 load 的其他指令也已经被执行了，处理起来就麻烦得多，需要执行一段补偿代码重新完成从 load 开始的所有前瞻指令，这时检测指令必须能够将控制流转移到补偿代码开始处。

6.2.5 开发更多的指令级并行

在谓词执行或前瞻执行等硬件机制的配合下，全局指令调度技术已经能够很好地处理由分支指令引起的控制相关：要么调度其他指令跨越那些比较容易预测的分支指令，进行前瞻执行；要么通过 if 转换删除那些难以预测的分支指令，进行谓词执行。此时，指令间的数据相关对指令级并行开发的限制作用反而越来越大。本节将讨论处理这些数据相关的编译优化技术，利用这些技术，编译时能够开发出更多的指令级并行。

本节所讨论的优化技术仍然是面向循环结构的，因为这种结构中蕴含着大量的并行性，始终是编译优化的重点对象。编译时，绝大多数指令级并行的分析和优化在中间代码或目标代码生成后才开始，面向中间代码或目标代码完成，但与循环结构相关的分析和优化通常需要在这之前进行，而且是面向源代码完成的。这主要是因为这些工作将首先识别出程序中的循环结构，并分析其中完成数组元素访问、数组元素下标计算等工作的指令。对编译器而言，在源代码上完成这些工作比在中间代码或目标代码上更加容易，因此本节的一些实例将针对 C 代码段进行分析。

1. 挖掘更多的循环级并行

从一个循环中究竟能够开发出多少并行受到两方面因素的制约：①同一循环迭代内部的指令相关，这属于基本指令调度问题，可以结合循环展开技术解决；②来自不同循环迭代的指令之间的相关，它会大大限制循环展开的效果。下面讨论处理这种相关的编译优化技术。

1) 循环携带相关

循环携带相关(loop-carried dependence)是限制循环结构并行性开发的一个重要因素，它是指一个循环的某个迭代中的指令与其他迭代中的指令之间的数据相关。

我们来看一个例子。在下面的循环中，

```
for(i = 1; i <= 100; i = i + 1){
    A[i + 1] = A[i] + C[i];              /* S1 */
    B[i + 1] = B[i] + A[i + 1];          /* S2 */
}
```

假设数组A、B和C中所有元素的存储地址都互不相同，请问语句S1与S2之间存在哪些数据相关?

S1和S2之间存在以下两种不同类型的数据相关。

(1) A[i+1]是第i次迭代中语句S1的目的操作数，它的源操作数A[i]是上一次迭代中语句S1的目的操作数，这是典型的RAW数据相关。由于引起数据相关的两条指令分属于不同的循环迭代，所以这是循环携带相关。相邻两次迭代中的语句S2之间也存在同样的数据相关。

(2) 同一循环迭代内的语句S2与S1之间也存在RAW数据相关，因为S2的源操作数A[i+1]恰好是S1的目的操作数。

尽管这两类数据相关都是RAW类型，但它们对于开发指令级并行的影响完全不同。下面是上例中的循环被展开两次后的结果，从中可以看出这些区别。循环携带相关迫使指令只能按照所在迭代的先后顺序依次执行，例如，S3无法被调度到S1之前，S4无法被调度到S2之前，因为这两组语句均存在循环携带相关。而迭代内各语句间的数据相关对于各次迭代是否能够并行没有任何影响，只要保证同一迭代内存在数据相关的各语句之间的相对顺序不变，多个循环迭代就可以并行执行。

```
for(i = 1; i <= 100; i = i + 2){
    A[i + 1] = A[i] + C[i];              /* S1,来自第1个迭代 */
    B[i + 1] = B[i] + A[i + 1];          /* S2,来自第1个迭代 */
    A[i + 2] = A[i + 1] + C[i + 1];      /* S3,来自第2个迭代 */
    B[i + 2] = B[i + 1] + A[i + 2];      /* S4,来自第2个迭代 */
}
```

可见，循环携带相关是制约循环级并行开发的主要因素。所谓循环级并行(loop level parallelism)是指循环中不同迭代之间的并行，它的粒度比指令级并行更粗，因为每个循环迭代中含有多条指令。循环级并行的存在意味着可以开发出更多的指令级并行。为了开发更多的循环级并行，必须想办法消除不同迭代之间的数据携带相关。

一般说来，相关距离越大，循环展开后能开发出的指令级并行就越多。

2) 存储别名分析

下面来讨论优化编译器对存储别名的分析和处理方法。

目前，基本上所有的存储别名分析算法都假设数组下标是仿射的。简单地说，如果一个一维数组$A[m:n]$的下标可以被表示为形如$a\times i+b$的形式，那么就称该数组是**仿射的**(affine)，这里i是循环索引变量，而m和n分别表示i取值的下界和上界。进一步，对于一

个多维数组，如果它每一维的下标都是仿射的，那么它就是仿射的。下标为 $x[y[i]]$ 的数组是一个典型的非仿射数组。

这样，判断访问数组 $A[m:n]$ 的两条指令是否相关的问题就可以被转换为判断这两条指令所访问的数组元素是否具有相同的下标。假设用下标 $a\times j+b$ 将数组 A 的一个元素写入存储器，而用下标 $c\times k+d$ 将数组 A 的一个元素从存储器中读出，这里 j 和 k 都是循环索引变量，$m\leqslant j$，$k\leqslant n$。显然，当且仅当 $a\times j+b=c\times k+d$ 时，这两条指令将会访问数组 A 的同一个元素。但在编译时很难判断该等式是否成立，除非 a、b、c、d 的值都是已知的。若 a、b、c、d 的值都已知，则可以通过 GCD(Greatest Common Divisor，最大公因数)测试法检测是否存在存储别名。GCD 测试方法可简单地描述为，如果 GCD(c, a)可以整除(d-b)，那么有可能存在存储别名。

3）数据相关分析

除了检测指令之间是否存在数据相关外，编译器还会将识别出的数据相关进一步细分为真数据相关、输出相关和反相关等不同类型，以便利用不同的优化技术消除这些相关。常用的优化技术包括重命名、值传播、高度消减等。

重命名优化通过引入新的变量，能够消除所有这些输出相关和反相关。

值传播优化是通过将变量替换为已知的值或表达式以达到消除数据相关的目的。通过值传播优化，可以将存在相关的两条或多条指令合并为一条。请看下面的例子：

```
DADDUI    R1,R2,#4
DADDUI    R1,R1,#4
```

通过值传播优化，这两条指令可被合并为一条：

```
DADDUI    R1,R2,#8
```

对于多流出处理器，简单地减少指令数量并不一定能够真正缩短执行时间，只有缩短这些指令所构成的数据流图中关键路径的长度才能真正获得性能提升。**高度削减**(tree height reduction)是解决这一问题的有效方法之一。以下面 3 条指令为例：

```
ADD    R1,R2,R3    /* I1 */
ADD    R4,R1,R6    /* I2 */
ADD    R8,R4,R7    /* I3 */
```

指令 I2 与 I1、I3 与 I2 之间均分别存在真数据相关，因此执行这 3 条指令共需要 3 个周期，即这段代码的数据流图的关键路径长度为 3。将它们等价地转换为下面的代码后，尽管指令总数没有改变，但数据流图关键路径的长度却由 3 减少为 2，执行时间缩短了一个周期。

```
ADD    R1,R2,R3
ADD    R4,R6,R7
ADD    R8,R1,R4
```

2. 软流水

前面介绍的循环展开和 Trace 调度两种优化技术都是通过扩大基本块的体积来开发更

多的指令级并行。本节讨论另一种比较常用的循环优化技术——软流水。

软流水(software pipelining)技术的核心思想是从循环不同的迭代中抽取一部分指令(循环控制指令除外)拼成一个新的循环迭代,以便将同一迭代中的相关指令分布到不同的迭代中,或将不同迭代中的相关指令封装到同一迭代中。

软流水的工作原理如图6.1所示。

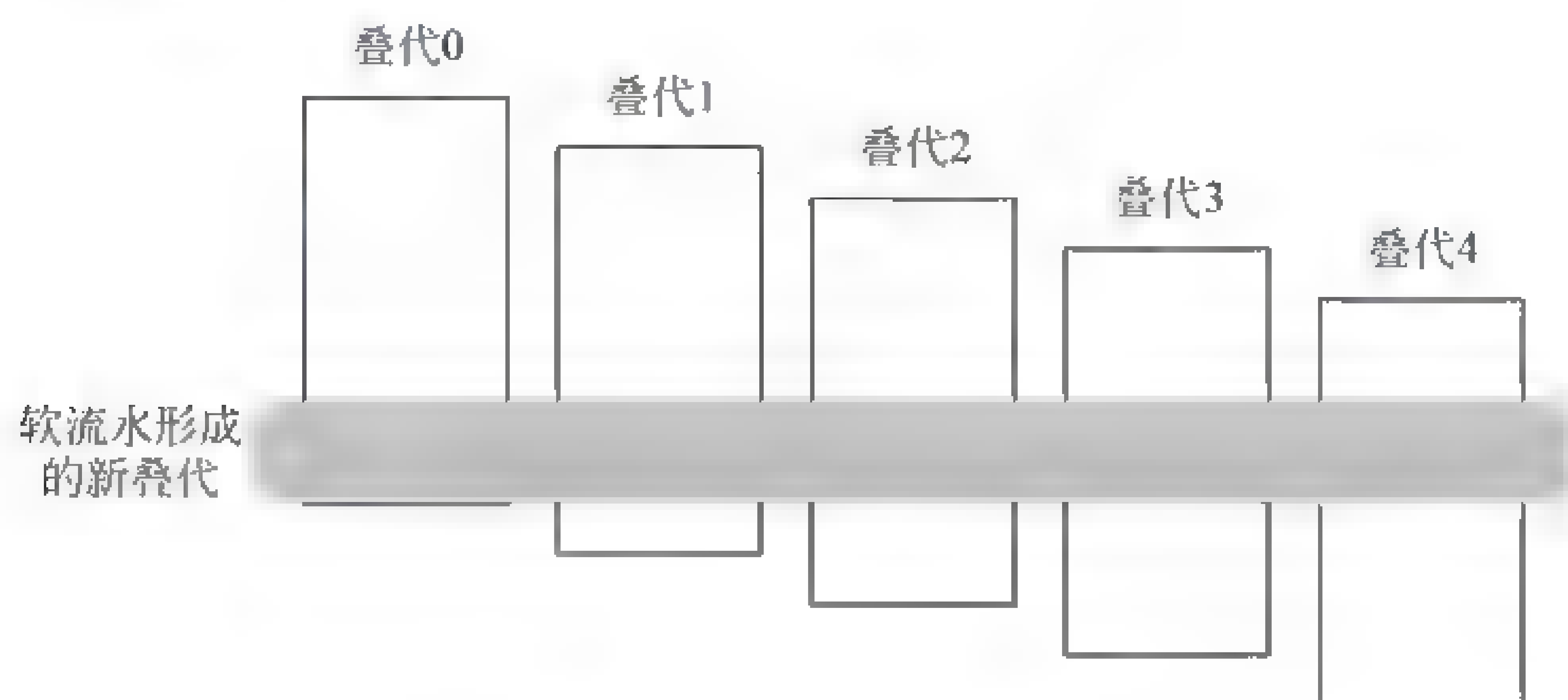


图6.1 软流水示意图

其中,阴影部分表示得到的新迭代,它包含原循环迭代0～4中的指令,执行这个新的迭代就同时执行了原循环中的迭代0～4,只不过每个迭代只有一部分指令被执行,而且各不相同,或者说每个迭代处于不同的执行阶段,就像在5级流水线中同时执行的5条指令一样。这项技术构造了一条虚拟的"流水线",以流水的方式同时执行循环中的多个不同迭代,这就是"软流水"这一名称的由来。与硬件流水线一样,这条软件"流水线"也需要一定时间才能充满(或排空),但与硬件流水线不同的是,用于充满(或排空)软件"流水线"的指令无法被封装到任何一个新的迭代中,只能放在新循环之前(或之后)。

软流水经常与循环展开结合在一起使用,除了需要从展开后的迭代中选择指令组成新的迭代外,对于那些迭代中有较多"空转"周期的循环,在进行软流水前往往也只有先进行循环展开才能获得比较理想的效果。不过,这两种技术所消除的流水线开销是完全不同的:循环展开主要减少由分支指令和修改循环索引变量的指令所引起的循环控制开销,如将某循环展开4次后,循环控制开销将减少为原来的四分之一,但执行每个迭代时,用于充满和排空流水线的开销并不会减少。软流水则恰恰减少了这部分开销,使迭代内的指令级并行达到最大。

习 题

1. 概念题

【题6.1】 解释下列名词

指令调度	循环展开	全局指令调度	关键路径
补偿代码	踪迹调度	超块	尾复制
VLIW	EPIC	二进制翻译	谓词执行

前瞻执行	if 转换	循环携带相关	相关距离
GCD 测试	值传播	高度削减	软流水

2. 问答题

【题 6.2】 简述踪迹调度和超块调度的基本过程。

【题 6.3】 试比较循环展开和软流水这两种编译优化技术的异同。

【题 6.4】 有哪些方法可以处理由前瞻指令引起的异常?

【题 6.5】 结合 IA-64 体系结构分析谓词执行机制需要编译器和流水线硬件分别提供哪些支持。

【题 6.6】 用 GCD 测试法判断下面的循环中是否存在循环携带的真数据相关。

```
for  (  i=2;
i<=100;i+=2)
a[i]=a[i-1];
```

【题 6.7】 试分析造成 VLIW 目标代码编码效率低的原因。如何解决这一问题?

3. 应用题

【题 6.8】 指出下面这段 C 代码中的所有数据相关,对于真数据相关,还应指明它们是否是循环携带相关。试分析能否从这段循环中开发出循环级并行,并说明原因。

```
for(i=2; i<100; i=i+1){
    a[i] = b[i] + a[i];              /* S1 */
    c[i-1] = a[i] + d[i];            /* S2 */
    a[i-1] = 2 * b[i];               /* S3 */
    b[i+1] = 2 * b[i];               /* S4 */
}
```

【题 6.9】 下面的循环中含有相关距离为 1 的循环携带相关,无法从中开发出大量的循环级并行。对这段代码进行调整,使得多个的循环迭代可以并行执行。

```
for(i=1; i<100; i=i+1){
    a[i] = b[i] + c[i];              /* S1 */
    b[i] = a[i] + d[i];              /* S2 */
    a[i+1] = a[i] + e[i];            /* S3 */
}
```

【题 6.10】 假设浮点流水线的延迟如表 6.2 所示。

表 6.2　浮点流水线的延迟

产生结果的指令	使用结果的指令	延迟(时钟周期数)
浮点计算	另一个浮点计算	3
浮点计算	浮点 store(S.D)	2
浮点 load(L.D)	浮点计算	1
浮点 load(L.D)	浮点 store(S.D)	0

```
Loop:   L.D      F0, 0(R1)          //取一个数组元素放入 F0
        ADD.D    F4, F0, F2         //加上在 F2 中的标量
        S.D      F4, 0(R1)          //存结果
        DADDIU   R1, R1, # -8       //将指针减少 8(每个数据占 8 个字节)
        BNE      R1, R2, Loop       //若 R1 不等于 R2,表示尚未结束,
                                    //转移到 Loop 继续
```

先将上述代码的循环展开3次得到4个循环体,然后对展开后的指令序列进行调度,消除所有的"空转"周期。假定R1的初值为32的倍数,即循环次数为4的倍数。消除冗余的指令,并且不要重复使用寄存器。

【题6.11】 下面是这段循环完成点积(dot product)运算,寄存器F2的初值为0。试结合使用循环展开和基本指令调度技术,消除其中的所有流水线"空转"周期。假设流水线延迟如表6.1所示,分支指令也会带来一个"空转"周期。

```
loop:   L.D      F0,0(R1)
        L.D      F4,0(R2)
        MUL.D    F0,F0,F4
        ADD.D    F2,F0,F2
        DADDUI   R1,R1,# -8
        DADDUI   R2,R2,# -8
        BNE      R1,R3,loop
```

【题6.12】 举例说明为何前瞻执行能够带来一定的性能提升?

【题6.13】 请描述下面的代码段中前瞻指令sLD如何执行。

```
        LD      R1,0(R3)         //取 A
        sLD     R14,0(R2)        //取 B,前瞻执行
        BEQZ    R1,L3
        DADDI   R14,R1,#4        //A = A + 4
L3:     SD      R14,0(R3)        //A = B
```

【题6.14】 试用软流水技术处理下面的循环。

```
L:   L.D      F0,0(R1)        //F0 = 0(R1)
     ADD.D    F4,F0,F2        //F2 = F0 + F1
     S.D      F4,0(R1)        //将 F2 存入 0(R1)
     DADDUI   R1,R1,#4        //R1 = R1 - 4
     BNEZ     R1,L            //若 R1 不为 0,转 L
```

【题6.15】 为了使用GCD测试方法判断一个循环是否含有存储别名,必须首先将循环索引变量的步长变为1。请先将下面的循环代码的步长变为1,然后用GCD测试方法判断其中是否存在存储别名。

```
for(i = 2; i <= 100; i += 2)
    a[i] = a[50 * i + 1];
```

【题6.16】 试将基本块B1的第二条语句调度至基本块B2内,为了保证结果正确,可以加入一定的补偿代码。

```
B1: x = x + 1
```

```
    y = x - y
    if x<5 goto B3
B2: z = x * z
    y = y + 1
    goto B5
B3: y = 2 * y
    x = x - 2
```

【题 6.17】 假设浮点流水线的延迟如表 6.3 所示。

表 6.3　浮点流水线的延迟

产生结果的指令	使用结果的指令	延迟(时钟周期数)
浮点计算	另一个浮点计算	3
浮点计算	浮点 store(S.D)	2
浮点 load(L.D)	浮点计算	1
浮点 load(L.D)	浮点 store(S.D)	0

假设某 VLIW 处理器每个时钟周期可以同时流出 5 个操作，包括两个访存操作，两个浮点操作以及一个整数或分支操作。将下述代码循环展开，并调度到该 VLIW 处理器上执行。循环展开次数不定，但至少要能够保证消除所有流水线“空转”周期，同时不考虑分支延迟。

```
Loop:  L.D     F0, 0(R1)        //取一个向量元素放入 F0
       ADD.D   F4, F0, F2       //加上在 F2 中的标量
       S.D     F4, 0(R1)        //存结果
       DADDIU  R1, R1, # -8     //将指针减 8(每个数据占 8 个字节)
       BNE     R1, R2, Loop     //若 R1 不等于 R2,表示尚未结束,转移
                                //到 Loop 继续
```

【题 6.18】 在下面的语句中，

```
if(A=0){S=T; }
```

假设变量 A、S、T 的值分别保存在寄存器 R1、R2 和 R3 内。请用分支指令和条件传输指令编写功能相同的汇编代码。

【题 6.19】 假设在一个周期内，某双流出的超标量处理器可以同时执行一个访存操作和一个 ALU 操作，或者仅执行一个分支操作。受此限制，表 6.4 中这段汇编代码的执行效率并不高，表现在：①第二个周期只能流出一条 ALU 指令，访存单元空闲；②当分支转移不成功时，BEQZ 指令后的两条 LW 指令之间存在的数据相关将引起流水线暂停。试通过谓词执行机制解决这两个问题，减少此段代码的执行开销。

表 6.4　汇编代码

周期	指令 1	指令 2
1	LW　R1,40(R2)	ADD　R3,R4,R5
2		ADD　R6,R3,R7
3	BEQZ　R10,L	
4	LW　R8,0(R10)	
5	LW　R9,0(R8)	

题　　解

1. 概念题

【题 6.1】 解释下列名词

指令调度——为了充分发挥流水线的作用,必须设法让它满负荷地工作,这就要求充分开发指令之间存在的并行性,找出不相关的指令序列,让它们在流水线上重叠并行执行,这一工作就是指令调度。

循环展开——所谓循环展开(loop unrolling)就是指把循环体的代码复制多次并按顺序排放,然后相应调整循环的结束条件。通过循环展开,多个循环迭代的代码可以合到一起调度,给编译器进行指令调度带来了更大的空间,而且还能够消除中间的分支指令和循环控制指令引起的开销。

全局指令调度——需要在多个基本块间移动指令的指令调度被称为"全局指令调度"。

关键路径——所谓关键路径(critical path),是指根据指令间相关关系构成的数据流图中延迟最长的一条路径。

补偿代码——在全局指令调度中,为保证结果正确而增加的一些代码。

踪迹调度——踪迹(trace)是程序执行的指令序列,通常由一个或多个基本块组成,trace 内可以有分支,但一定不能包含循环。踪迹调度会优化执行频率高的 trace,减少其执行开销。踪迹调度包括踪迹选择和踪迹压缩两个步骤。

超块——人们增加了对 trace 拓扑结构的约束,将其限制为只能拥有一个入口,但可以拥有多个出口的结构,这种新结构被称为超块(superblock)。

尾复制——超块调度时往往也需要进行代码复制,以确保结果的正确。由于被复制的代码段总是作为退出超块后必须执行的补偿代码,故这种技术被形象地称为尾复制(tail duplication)。

VLIW——VLIW 能够把同时流出的或者满足特定约束的一组操作打包在一起,得到一条更长(64 位、128 位或更长)的指令,这就是其名字的由来。每个操作被放在 VLIW 指令的一个槽(slot)内。VLIW 处理器执行这样一条长指令就相当于超标量处理器同时执行多条指令,从而实现了多流出。

EPIC——一般来说,EPIC 结构必须符合以下两个基本特点。第一,指令级并行主要由编译器负责开发,处理器应为保证代码正确执行提供必要的硬件支持,只有在这些硬件机制的辅助下这些优化技术才能高效完成。第二,系统结构必须提供某种通信机制,使得流水线硬件能够了解编译器"安排"好的指令执行顺序。EPIC 并不仅仅是采用了多种高级编译优化技术的 VLIW 结构,这只不过是它的一个特征。EPIC 的第二个特征,有效的软硬件通信机制,才是它与 VLIW 之间的本质区别。

二进制翻译——二进制翻译(binary translation)是指将某个硬件平台的二进制目标代码翻译为另一个平台的目标代码的过程,是目前在不同平台之间实现目标代码兼容的主要手段之一。

谓词执行——谓词执行(predicated execution)是一种特殊的条件执行机制。所谓条件执行,是指指令的执行依赖于一定的条件,当条件为真时指令将正常执行,否则指令将被动态地转换为空操作。

前瞻执行——前瞻(speculation)执行并不仅仅是在数据相关或控制相关尚未消除的情况下,简单地将指令调度到与之相关的指令之前猜测执行,它还通过一系列复杂的硬件机制完成异常处理、正确性保证等工作。

if 转换——将分支指令引起的控制相关转换为相对于分支转移条件的数据相关,使得那些本应在流水线前端 ID 段处理的控制相关被推迟到流水线后端 WB 段处理,为指令调度提供了更大的空间。而且,借助这种转换可以删除代码中那些行为难以预测的分支指令,提高分支预测准确率,并减少由于分支预测错误带来的性能损失。这种转换技术称作 if 转换。

循环携带相关——循环携带相关(loop-carried dependence)是限制循环结构并行性开发的一个重要因素,它是指一个循环的某个迭代中的指令与其他迭代中的指令之间的数据相关。

相关距离——一个循环的相关距离是 k,表示它的第 i 次迭代中的语句与第 i-k 次迭代中的语句之间存在循环携带数据相关。

GCD 测试——这是一种存储别名检测方法,可简单地描述为,如果 GCD(c, a)可以整除(d-b),那么有可能存在存储别名。

值传播——值传播优化是通过将变量替换为已知的值或表达式以达到消除数据相关的目的。

高度削减——通过代码调度/转换减少数据流图关键路径长度的一种优化方法。

软流水——软流水(software pipelining)技术的核心思想是从循环不同的迭代中抽取一部分指令(循环控制指令除外)拼成一个新的循环迭代,以便将同一迭代中的相关指令分布到不同的迭代中,或将不同迭代中的相关指令封装到同一迭代中。

2. 问答题

【题 6.2】 答:踪迹调度过程分为两步。第一步称为踪迹选择(trace selection),负责从程序的控制流图中选择执行频率较高的路径,每条路径就是一条 trace。第二步称为踪迹压缩(trace compaction),即对已生成的 trace 进行指令调度和优化,尽可能地缩短其执行时间。在踪迹调度中,由于选出的 trace 都是执行频率很高的路径,减少它们的执行开销有助于缩短程序的总执行时间,这就是踪迹调度能够提升性能的最根本原因。

超块是只能拥有一个入口,但可以拥有多个出口的 trace 结构,可被视作一种扩展的基本块结构,其构造过程与 trace 非常相似。但与踪迹调度相比,超块调度可以简化补偿代码生成过程,并降低指令调度的复杂度,但由于其结构的限制(只有一个入口),超块结构的目标代码体积也会大大增加。

【题 6.3】 答:它们都可以开发循环级并行,即不同迭代的指令之间的并行。但是,循环展开增加了代码体积,减少了循环次数;而软流水对代码体积和循环次数的影响都很小。

这两种技术所消除的流水线开销是完全不同的:循环展开主要减少由分支指令和修改循环索引变量的指令所引起的循环控制开销,如将某循环展开 4 次后,循环控制开销将减少

为原来的四分之一,但执行每个迭代时,用于充满和排空流水线的开销并不会减少。软流水则恰恰减少了这部分开销,使迭代内的指令级并行达到最大。

因此它们往往被结合在一起使用。

【题6.4】 答:通常有4种方法处理前瞻指令引起的异常:

(1) 立即处理,不管它们是否由被前瞻执行的指令引发。

(2) 借助专门的检测指令判断是否需要进行异常处理。

(3) poison 位方法。

(4) 基于硬件的前瞻执行机制,待确认前瞻执行正确后才处理。

【题6.5】 答:第一,提供谓词寄存器。IA-64设置了大量的谓词寄存器(64个),用于保存指令的执行条件,每个谓词寄存器的宽度都是1位。

第二,修改指令格式。所有类型的IA-64操作都可以按照谓词执行方式执行,每个操作41位编码的最低6位指明了保存执行条件的谓词寄存器。

第三,增加专门的谓词计算指令。为了更高效地确定指令的执行条件(谓词寄存器的值),IA-64专门增强了比较(compare)操作和测试(test)操作的功能。它们都可以同时修改谓词寄存器的值,这样就可以同时计算出一个分支结构 then 部分和 else 部分各指令的谓词。IA-64的比较操作更是支持多种不同的比较模式,可以同时完成多个比较,大大地提高了谓词的计算效率。

第四,提供专门的异常处理机制。

第五,流水线硬件决定何时将执行条件不成立的谓词指令转换为空操作,可以有两种选择:流水线前端指令流出时,或是流水线后端结果确认时。

第六,提供能够完成 if-conversion 的编译器。

【题6.6】 答:在这个循环中 $a=1$, $b=0$, $c=1$, $d=-1$,这样,$GCD(a,c)=1$, $d-b=-1$,由于前者可以整除后者,该循环存在循环携带的真数据相关,相关距离为1。

【题6.7】 答:造成VLIW目标代码编码效率低的原因主要有两个:①为了消除流水线"空转"需要增加循环展开的次数,这增加了目标代码的体积;②很难从应用程序中找到足够多的并行指令填满VLIW指令中的每一个槽,实际应用中还经常会出现一条VLIW指令中所有操作都是nop的情况。

使用代码压缩/还原技术可以减少VLIW目标代码体积过大带来的性能损失:VLIW目标代码被压缩后保存在硬盘中,只保留指令中非nop操作的信息;程序运行时,当代码被加载到指令Cache时或译码时再将其解压缩,还原出所含的nop操作。

3. 应用题

【题6.8】

解:

(1) 因为对a[i]的写和读,S1与S2之间存在真数据相关,不是循环携带相关。它不会妨碍我们将这段循环展开,但却限制S2不能被调度到S1之前执行。

(2) 因为对a[i]的写和读,S3与S1之间存在真数据相关,是循环携带相关。

(3) 因为对b[i]的写和读,S4与S1、S4与S3、S4与S4之间存在真数据相关,是循环携带相关,且相关距离为1,这使我们无法有效地开发循环级并行。

(4) 因为对 a[i]的写,S1 和 S3 之间存在输出相关,是循环携带相关。

【题 6.9】

解:第 i 次迭代中语句 S3 的结果被第 i+1 次迭代中语句 S1 所使用,S1 与 S3 之间存在循环携带真数据相关。但它们没有构成环,因此多个循环迭代之间仍可并行,但需要进行如下代码变换:

```
a[1] = b[1] + c[1];
a[2] = a[1] + e[1];
for (i=1; i<99; i=i+1) {
    b[i] = a[i] + d[i];
    a[i+1] = b[i+1] + d[i+1];
    a[i+2] = a[i+1] + e[i+1];
}
b[99] = a[99] + d[99];
```

【题 6.10】

解:展开后没有调度的代码如下。

```
Loop:  L.D     F0, 0(R1)
       ADD.D   F4, F0, F2
       S.D     F4, 0(R1)
       L.D     F6, -8(R1)
       ADD.D   F8, F6, F2
       S.D     F8, -8(R1)
       L.D     F10, -16(R1)
       ADD.D   F12, F10, F2
       S.D     F12, -16(R1)
       L.D     F14, -24(R1)
       ADD.D   F16, F14, F2
       S.D     F16, -24(R1)
       DADDIU  R1, R1, # -32
       BNE     R1, R2, Loop
```

对指令序列进行优化调度,以减少空转周期。

```
                                     指令流出时钟
Loop:  L.D       F0, 0(R1)              1
       L.D       F6, -8(R1)             2
       L.D       F10, -16(R1)           3
       L.D       F14, -24(R1)           4
       ADD.D     F4, F0, F2             5
       ADD.D     F8, F6, F2             6
       ADD.D     F12, F10, F2           7
       ADD.D     F16, F14, F2           8
       S.D       F4, 0(R1)              9
       S.D       F8, -8(R1)             10
       DADDIU    R1, R1, # -32          12
       S.D       F12, 16(R1)            11
       BNE       R1, R2, Loop           13
       S.D       F16, 8(R1)             14
```

这个循环由于没有数据相关引起的空转等待，整个循环仅使用了14个时钟周期，平均每个元素使用14/4=3.5个时钟周期。

【题6.11】

解：

```
loop:  L.D     F0,16(R1)
       L.D     F4,16(R2)
       L.D     F6,8(R1)
       MUL.D   F0,F0,F4
       L.D     F8,8(R2)
       L.D     F10,F10,(R1)
       MUL.D   F6,F6,F8
       ADD.D   F2,F0,F2
       L.D     F12,F12,(R2)
       DADDUI  R1,R1,# -24
       MUL.D   F10,F10,F12
       ADD.D   F2,F6,F2
       DADDUI  R2,R2,# -24
       BNE     R1,R3,loop
       ADD.D   F2,F10,F2
```

【题6.12】

解：以下面的汇编指令为例：

```
      LD     R1,0(R3)          //取 A
      BNEZ   R1,L1             //(A≠0)?
      LD     R1,0(R2)          //A = B(else 部分)
      J      L2
L1:   DADDI  R1,R1,#4          //A = A + 4(then 部分)
L2:   SD     R1,0(R3)          //存 A
```

前瞻调度结果如下：

```
      LD     R1,0(R3)          //取 A
      SLD    R14,0(R2)         //取 B,前瞻执行
      BEQZ   R1,L3
      DADDI  R14,R1,#4         //A = A + 4
L3:   SD     R14  ,0(R3)       //A = B
```

这段代码将按照下面的顺序执行：第一条指令正常执行；第二条指令为前瞻执行，此处我们特地用SLD表明它与第一条指令的区别，取出的数据保存在寄存器R14中；第三条是分支指令BEQZ，它将根据分支转移条件(即R1是否为0)判断前瞻是否正确，若R1为0则跳转到L3并执行指令SD，由于此时变量B的值已经被加载到R14中，通过这条指令就会将变量B的值赋给A，此时前瞻正确；否则投机失败，应先执行第四条指令DADDI，将R1的值加4，然后通过下一条指令SD将结果写回变量A对应的存储单元。

【题6.13】

解：这段代码将按照下面的顺序执行：第一条指令正常执行；第二条指令为前瞻执行，取出的数据保存在寄存器R14中；第三条是分支指令BEQZ，它将根据分支转移条件(即

R1 是否为 0)判断前瞻是否正确,若 R1 为 0 则跳转到 L3 并执行指令 SD,由于此时变量 B 的值已经被加载到 R14 中,通过这条指令就会将变量 B 的值赋给 A,此时前瞻正确;否则前瞻失败,应先执行第四条指令 DADDI,将 R1 的值加 4,然后通过下一条指令 SD 将结果写回变量 A 对应的存储单元。

【题 6.14】

解:软流水处理后结果如下:

```
    DADDUI    R1,R1,# -8        //R1 保存 x[n-2]的地址
    L.D       F0,8(R1)          //取 x[n]
    ADD.D     F4,F0,F2          //x[n] = x[n] + F2
    L.D       F0,4(R1)          //取 x[n-1]
L:  S.D       F4,8(R1)          //存 x[i+2]
    ADD.D     F4,F0,F2          //x[i+1] = x[i+1] + F2
    L.D       F0,0(R1)          //取 x[i]
    DADDUI    R1,R1,# -4        //填充分支延迟槽
    BNE       R1,0,Loop
    S.D       F0,4(R1)          //存 x[2]
    ADD.D     F4,F0,F2          //x[1] = x[1] + F2
    S.D       F4,0(R1)          //存 x[1]
```

【题 6.15】

解:修改结果如下:

```
for(i=1; i<=50; i++)
    a[2*i] = a[100*i+1];
```

用 GCD 测试法,a=2,b=0,c=100,d=1,GCD(c,a)=2,(d-b)=1,因此不存在存储别名。

【题 6.16】

解:

```
B1: x = x + 1
    if x<5 goto B3
B2: y = x - y
    z = x * z
    y = y + 1
    goto B5
B3: y = x - y         //补偿代码,若分支转移成功,需要这条代码,否则 y 的值将不正确
    y = 2 * y
    x = x - 2
```

【题 6.17】

解:循环被展开 7 次,经调度后可以消除所有流水线"空转",如表 6.5 所示。

表 6.5　循环展开和指令调度

访存操作 1	访存操作 2	浮点操作 1	浮点操作 2	整数分支操作
L. D F0,0(R1)	L. D F6,−8(R1)	nop	nop	nop
L. D F10,−16(R1)	L. D F14,−24(R1)	nop	nop	nop
L. D F18,−32(R1)	L. D F22,−40(R1)	ADD. D F4,F0,F2	ADD. D F8,F6,F2	nop

续表

访存操作1	访存操作2	浮点操作1	浮点操作2	整数分支操作
L. D F26,-48(R1)	nop	ADD. D F12,F10,F2	ADD. D F16,F14,F2	nop
nop	nop	ADD. D F20,F18,F2	ADD. D F24,F22,F2	nop
S. D 0(R1),F4	S. D -8(R1),F8	ADD. D F28,F26,F2	nop	nop
S. D -16(R1),F12	S. D -24(R1),F16	nop	nop	nop
S. D -32(R1),F20	S. D -40(R1),F24	nop	nop	DADDUI R1,R1,#56
S. D 8(R1),F28	nop	nop	nop	BNE R1,R2,Loop

【题 6.18】

解：包含分支指令的MIPS汇编代码如下：

```
    BNEZ    R1,L
    ADDU    R2,R3,R0
L:
```

而使用条件传输指令时的汇编代码为：

```
CMOVZ    R2,R3,R1
```

指令CMOVZ有3个操作数，R2为目的操作数，R1和R3是源操作数，执行条件保存在寄存器R1中。当R1=0时，R3的值被复制到R2中，否则R2的内容不变。

【题 6.19】

解：用LWC表示带谓词的LW指令，并假设该指令的执行条件为谓词不等于0。这样，BEQZ后的第一条LW指令就可以被转换为LWC指令，并被调度到第二个周期执行，如表6.6所示。

表 6.6 调度后代码

周期	指令1	指令2
1	LW R1,40(R2)	ADD R3,R4,R5
2	LWC R8,20(R10),R10	ADD R6,R3,R7
3	BEQZ R10,L	
4	LW R9,0(R8)	

显然调度后代码的执行时间缩短了。当然，如果分支转移成功(即R10=0)，LWC指令将被转换为空操作，这虽然不会影响结果的正确性，但也不会缩短这段代码的执行时间。

第 7 章　存储系统

7.1　基本要求与难点

7.1.1　基本要求

(1) 掌握有关存储系统的基本概念。
(2) 理解存储系统的层次结构和存储层次的性能参数。
(3) 掌握"Cache-主存"和"主存-辅存"层次的区别。
(4) 理解 Cache 的基本工作原理和工作过程，掌握映像规则。
(5) 掌握替换算法及 LRU 算法的硬件实现，并能进行 Cache 性能分析。
(6) 掌握 8 种降低 Cache 不命中率的方法。
(7) 掌握 5 种减少 Cache 不命中开销的方法。
(8) 掌握 4 种减少 Cache 命中时间的方法。
(9) 理解并行主存系统，掌握单体多字并行存储器、多体交叉存储器以及避免存储体冲突的方法。
(10) 掌握虚拟存储器中的快速地址转换技术。
(11) 了解 AMD Opteron 的存储器层次结构。

7.1.2　难点

(1) Cache 的基本工作原理和映像规则。
(2) Cache 性能分析。
(3) 虚拟 Cache。
(4) 并行主存系统。

7.2　知识要点

7.2.1　存储系统的层次结构

1. 存储系统的层次结构

从实现"容量大、价格低"的要求来看，应采用能提供大容量的存储器技术；但从满足性

能需求的角度来看，又应采用昂贵且容量较小的快速存储器。走出这种困境的唯一方法，是采用多种存储器技术，构成多级存储层次结构。

1）程序访问的局部性原理

程序访问的**局部性原理**是指：对于绝大多数程序来说，程序所访问的指令和数据在地址上不是均匀分布的，而是相对簇聚的。它是构成存储层次的基础原理。

2）存储系统的多级层次结构

存储系统的多级层次结构如图7.1所示。其中，$M_1, M_2, \cdots, M_n$是用不同技术实现的存储器。假设第i个存储器M_i的访问时间为T_i，容量为S_i，平均每位价格为C_i，则该存储系统中n个存储器的参数满足以下关系：

访问时间：　$T_1 < T_2 < \cdots < T_n$

容量：　$S_1 < S_2 < \cdots < S_n$

平均每位价格：$C_1 > C_2 > \cdots > C_n$

整个存储系统要达到的目标是：从CPU来看，该存储系统的速度接近于M_1的，而容量和每位价格都接近于M_n的。

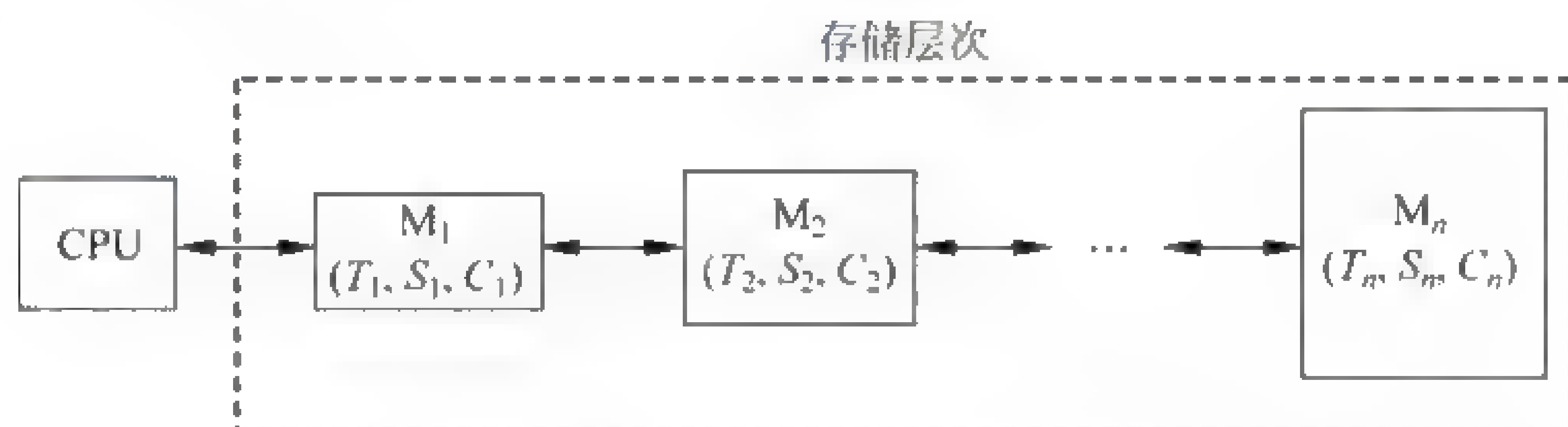


图7.1　存储系统的层次结构

在存储层次中，各存储器之间一般满足包容关系，即任何一层存储器中的内容都是其下一层(离CPU更远的一层)存储器中内容的子集。CPU与M_1之间传送信息一般是以字为单位，M_1以外(含M_1)的相邻存储器之间一般以块或页面为单位传送信息。

2. 存储系统的性能参数

为简单起见，我们仅考虑由M_1和M_2两个存储器构成的两级存储层次结构，M_1的容量、访问时间和每位价格分别为S_1, T_1, C_1；M_2的参数为S_2, T_2, C_2。

1）存储容量S

一般来说，整个存储系统的容量即是第二级存储器M_2的容量，即$S=S_2$。

2）存储系统的平均每位价格C

$$C = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$$

显然，当$S_1 << S_2$时，$C \approx C_2$。

3）命中率H

命中率H的定义为：CPU访问该存储系统时，在M_1中找到所需信息的概率。其计算公式为：

$$H = \frac{N_1}{N_1 + N_2}$$

其中，N_1 和 N_2 分别为访问 M_1 和 M_2 的次数，可以通过模拟执行一组有代表性的程序来获得。

不命中率：$F=1-H$

4)平均访存时间 T_A

$$T_A=HT_1+(1-H)(T_1+T_M)=T_1+(1-H)T_M$$

或 $T_A=T_1+FT_M$

其中，$T_M=T_2+T_B$，常称为**不命中开销**，T_1 常称为**命中时间**。

T_B 为传送一个信息块所需的时间。

3. 三级存储系统

目前，大多数计算机都采用了由 Cache(高速缓冲存储器)、主存储器和磁盘存储器(辅存)构成的三级存储系统。这个存储系统可以看成是由"Cache-主存"层次和"主存-辅存"层次构成的系统。

1) "Cache-主存"层次

近十多年，CPU 的性能提高得很快，但主存性能的提高却慢得多。因此，CPU 和主存之间在速度上的差距越来越大。为了填补这个差距，现代计算机都是在 CPU 和主存之间设置一个高速缓冲存储器 Cache。这个 Cache 对于提高整个计算机系统的性能有着重要的意义，几乎是一个不可缺少的部件。一般来说，这个层次的工作完全由硬件实现，所以它不但对应用程序员是透明的，而且对系统程序员也是透明的。

2) "主存-辅存"层次

这个层次的目的是为了弥补主存容量的不足。它是在主存外面增设一个容量更大、每位价格更低、但速度更慢的存储器(称为辅存，一般是硬盘)。它们依靠辅助软硬件的作用，构成一个整体。"主存-辅存"层次常被用来实现虚拟存储器，向编程人员提供"用不完"的程序空间。

3) 两者的比较

表 7.1 对"Cache-主存"和"主存-辅存"层次做了一个简单的比较。

表 7.1　"Cache-主存"与"主存-辅存"层次的区别

比较项目 \ 存储层次	"Cache-主存"层次	"主存-辅存"层次
目的	为了弥补主存速度的不足	为了弥补主存容量的不足
存储管理的实现	全部由专用硬件实现	主要由软件实现
访问速度的比值(第一级比第二级)	几比一	几万比一
典型的块(页)大小	几十个字节	几百到几千个字节
CPU 对第二级的访问方式	可直接访问	均通过第一级
不命中时 CPU 是否切换	不切换	切换到其他进程

4. 存储层次的 4 个问题

下面将展开论述"Cache-主存"层次。首先要考虑以下 4 个问题。

(1) 当把一个块(页)调入高一层(靠近CPU)存储器时,可以放到哪些位置上?(映像规则)

(2) 当所要访问的块(页)在高一层存储器中时,如何找到该块?(查找算法)

(3) 当发生不命中而且高一层存储器已经满时,应替换哪一块?(替换算法)

(4) 当进行写访问时,应进行哪些操作?(写策略)

搞清楚这些问题,对于理解一个具体存储层次的工作原理是十分重要的。

7.2.2 Cache基本知识

1. 基本结构和原理

略。

2. 映像规则

当要把一个块从主存调入Cache时,可以放置到哪些位置?这就是映像规则所要解决的。映像规则有以下3种。

1) 全相联映像

全相联是指主存中的任一块可以被放置到Cache中的任意一个位置。

2) 直接映像

直接映像是指主存中的每一个块只能被放置到Cache中唯一的一个位置。从主存块到Cache块的对应关系是依次循环分配的。

一般地,如果主存的第i块(即块地址为i)映像到Cache块的第j块,则:

$$j = i \bmod M$$

其中,M为Cache的块数。

设$M=2^m$,则当表示为二进制数时,j实际上就是i的低m位。

3) 组相联映像

在**组相联映像**中,Cache被等分为若干组,每组由若干个块构成。主存中的每一块可以被放置到Cache中唯一的一个组中的任何一个位置。它是直接映像和全相联的一种折中。组的选择常采用位选择算法,即对于主存的第i块,若它所映像到Cache组的组号为k,则有:

$$k = i \bmod G$$

其中,G为Cache的组数。

设$G=2^g$,则当表示为二进制数时,k实际上就是i的低g位。

这里的低g位以及上述直接映像中的低m位通常称为**索引**(index)。

如果每组中有n个块($n=M/G$),则称该映像规则为**n路组相联**。

相联度越高(即n的值越大),Cache空间的利用率就越高,块冲突概率就越低,因而Cache的不命中率就越低。**块冲突**是指当要把一个主存块调入Cache时,按映像规则所对应的Cache块位置都已经被占用。绝大多数计算机都采用直接映像、两路组相联或4路组相联。特别是直接映像,应用得最多。

3. 查找方法

Cache 中设有一个目录表，每一个 Cache 块在该表中都有唯一的一项，用于指出当前该块中存放的信息是哪个主存块的。它实际上是记录了该主存块的块地址的高位部分，称为**标识**(tag)。每个主存块能唯一地由其标识来确定。标识在主存块地址中的位置如图 7.2 所示。



图 7.2　主存地址的分割

由于目录表中存放的是标识，所以存放目录表的存储器又称为标识存储器。目录表中给每一项设置一个有效位，用该位为“1”表示 Cache 中相应块所包含的信息有效。

根据映像规则不同，一个主存块可能映像到 Cache 中的一个或多个 Cache 块位置。为便于讨论，我们把它们称为**候选位置**。当 CPU 访问该主存块时，必须且只须查找它的候选位置的目录表项(标识)。为了保证速度，对各候选位置的标识的检查应并行进行。

在采用直接映像或组相联映像的情况下，为了提高访问速度，一般是把“主存→Cache”地址变换和访问 Cache 存储体安排成同时进行。这时，由于还不知道哪个候选位置上有所要访问的数据，所以就把所有候选位置中的相应信息都读出来，在“主存→Cache”地址变换完成后，再根据其结果从这些信息中选一个(如果命中)，发送给 CPU。

直接映像 Cache 的候选位置最少，只有一个；全相联 Cache 的候选位置最多，为 M 个；而 n 路组相联则介于两者之间，为 n 个。实现并行查找的方法有两种：①用相联存储器实现；②用单体多字的按地址访问的存储器和比较器来实现。

采用第①种方法时，其目录由 2^g 个相联存储区构成，每个相联存储区的大小为 $n\times(h+\log_2 n)$(h 为标识的位的位数)，需用相联存储器来实现。查找时，是用索引从 2^g 个相联存储区找出相应的区，然后进行相联比较。找出其组内块地址。

采用第②种方法时，需要一个大小为 $2^g\times n\times h$ 位的按地址访问的存储器和 n 个 h 位的比较器。查找时，是用索引从 2^g 行中选出一行 n 个标识，然后将这些标识与本次访问的标识并行进行比较。并根据比较结果去选择相应的数据。当相联度 n 增加时，不仅比较器的个数会增加，而且比较器的位数也会增加。

4. Cache 的工作过程

下面以 DEC 的 Alpha AXP 21064 微处理器中的数据 Cache 为例，来介绍 Cache 的工作过程。图 7.3 为其结构框图。这是一个容量为 8KB 的直接映像 Cache，块大小为 32B，共有 256 个块。采用写直达法，写缓冲器的大小为 4 个块。

下面参照图 7.3 来介绍访问 Cache 的工作过程(图中带圈的数字表示步骤的顺序)。

21064 微处理器传送给 Cache 的物理地址为 34 位(图中的①)。这个地址被分为两部分：块地址(29 位)和块内位移(5 位)。块地址又进一步被分为标识和索引。

第②步是用索引作为地址从 256 个目录项中选择一项，读出相应的标识和有效位。同

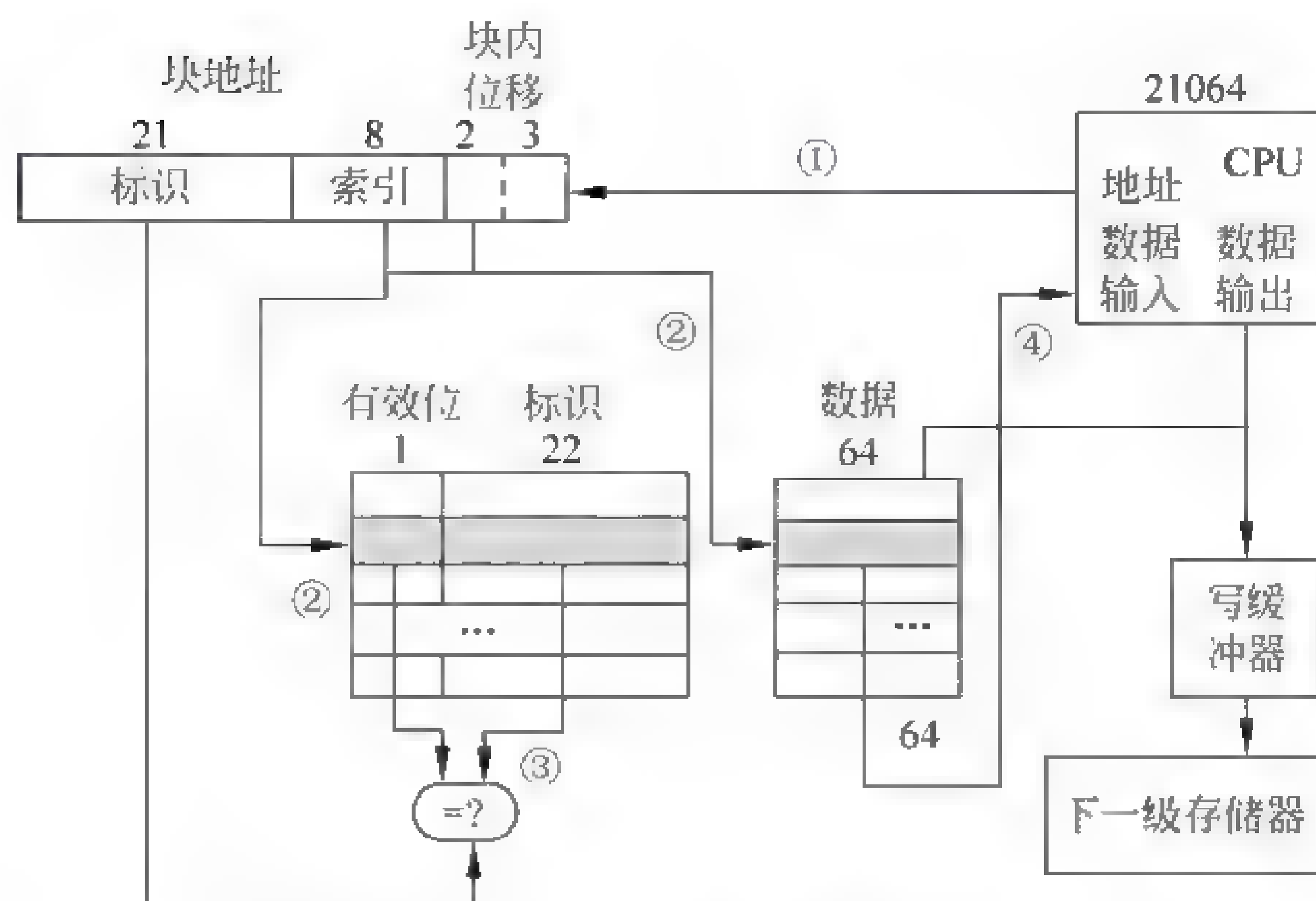


图 7.3　Alpha AXP 21064 微处理器中数据 Cache 的结构

时用索引作为地址从 Cache 的数据体中选择一块，用块内位移的高两位作为偏移量读出相应的数据字（每个块为 4 个字，每个字为 8 个字节）。在直接映像 Cache 中，"读出数据并送往 CPU"与"读出标识并进行匹配"这两个过程可以并行进行。

第③步是把上一步读出的标识与 CPU 送来的物理地址中的标识进行比较。如果标识比较的结果是匹配，且有效位为"1"，就表示本次访问 Cache 命中，那么最后一步（即第④步）就是发信号通知 CPU 取走数据。21064 完成这 4 步需要两个时钟周期。

当发生读不命中时，Cache 向 CPU 发出一个暂停信号，通知它等待，并从下一级存储器中新调入一个数据块。由于 21064 的数据 Cache 是直接映像的，所以当发生调块时，必然引起替换。

和任何 Cache 一样，在 21064 中对"写"的处理比对"读"的处理更复杂。在写命中的情况下，前三步跟上面是一样的。最后，在确认标识比较为匹配之后，才把数据写入。

不管是否命中，21064 都要把数据写入主存（即采用的是写直达法）。为了提高"写"访问的速度，21064 设置了一个写缓冲器。这个写缓冲器是按字寻址的，它含有 4 个块，每块大小为 4 个字。当要进行写入操作时，如果写缓冲器不满，那么就把数据和完整的地址写入缓冲器。对 CPU 而言，本次"写"访问已完成，CPU 可以继续往下执行。由写缓冲器负责把该数据写入主存。

5. 替换算法

1）替换算法

直接映像 Cache 中的替换很简单，因为只有一个块，别无选择。而在组相联和全相联 Cache 中，则有多个块供选择，我们当然希望应尽可能避免替换掉马上就要用到的信息。主要的替换算法有以下 3 种。

（1）随机法

这种方法随机地选择被替换的块。其优点是简单、易于用硬件实现，但这种方法没有考虑 Cache 块过去被使用的情况，反映不了程序的局部性，所以命中率比较低。

(2) 先进先出法(First-In-First-Out,FIFO)

这种方法选择最早调入的块作为被替换的块。其优点也是容易实现。它虽然利用了同一组中各块进入 Cache 的先后顺序这一"历史"信息,但还是不能正确地反映程序的局部性。因为最先进入的块,也可能是经常要用到的块。

(3) 最近最少使用法(Least Recently Used,LRU)

这种方法本来是选择近期使用次数最少的块作为被替换的块。但由于其实现比较复杂,现在实际上实现的 LRU 都只是选择最久没有被访问过的块(也称为 LFU 算法)。

LRU 能较好地反映程序的局部性原理,因而其命中率在上述 3 种方法中是最高的。但是 LRU 比较复杂,硬件实现成本比较高,特别是当组的大小增加时,LRU 的实现代价会越来越高。

LRU 和随机法分别因其不命中率低和实现简单而被广泛采用。不过,有模拟数据表明,对于容量很大的 Cache,LRU 和随机法的命中率差别不大。

2) LRU 算法的硬件实现

(1) 堆栈法

这种方法是用一个堆栈来记录组相联 Cache 的同一组中各块被访问的先后次序。这个先后次序是用堆栈元素的物理位置来反映的,从栈底到栈顶的先后次序依次记录了该组中各块被访问的先后次序(存放的是组内块地址)。栈顶记录的是刚访问过的块,次栈顶记录的是前一次刚被访问过的块……栈底记录的是最早被访问过的块。这样,当需要替换时,就可以从栈底得到应该被替换的块(块地址)。

显然,随着 Cache 访问的发生,该堆栈中的内容必须动态更新,以便保持该堆栈的上述属性。当 Cache 访问命中时,通过用块地址进行相联查找,在堆栈中找到相应的元素,然后把该元素的上面的所有元素下压一个位置,同时把本次访问的块地址抽出来,从最上面压入栈顶。而该元素下面的所有元素则保持不动。

如果 Cache 访问不命中,则把本次访问的块地址从最上面压入栈顶,堆栈中所有原来的元素都下移一个位置。如果 Cache 中该组已经没有空闲块,就要替换一个块。这时从栈底被挤出去的块地址就是需要被替换的块的块地址。

(2) 比较对法

比较对法的基本思路是让各块两两组合,构成比较对。每一个比较对用一个触发器的状态来表示它所相关的两个块最近一次被访问的远近次序,再经过门电路就可找到 LRU 块。例如,对于由 A、B、C 组成的组来说,用 T_{AB} 为"1"表示 A 比 B 更近被访问过;T_{AB} 为"0"表示 B 比 A 更近被访问过。T_{AC}、T_{BC} 也是按这样的规则定义。则有:

$$C_{LRU} = T_{AC} \cdot T_{BC}$$

同理可得:

$$B_{LRU} = T_{AB} \cdot T_{BC}$$

$$A_{LRU} = T_{AB} \cdot T_{AC}$$

所需要的触发器的个数与两两组合的比较对的数目相同。这是一个 P 中取 2 的组合问题:$C_P^2 = P\times(P-1)$。在块数少时,它所需要的硬件较少,比前面的堆栈法更容易实现。但随着组内块数 P 的增加,它所需的触发器的个数会以平方的关系迅速增加,门的输入端数也线性增加。当 P 超过 8 时,所需要的触发器个数就多得不能承受了。不过在这种情况

下可以用多级状态位技术来减少所需的硬件。

6. 写策略

按照存储层次的要求，Cache 内容应是主存部分内容的一个副本。但是“写”访问却可能导致它们内容的不一致。这就产生了 Cache 与主存内容的一致性问题。显然，为了保证正确性，主存的内容也必须更新。至于何时更新，这正是写策略所要解决的问题。

写策略是区分不同 Cache 设计方案的一个重要标志。写策略主要有以下两种。

(1) 写直达法。也称为存直达法。它是指在执行“写”操作时，不仅把数据写入 Cache 中相应的块，而且也写入下一级存储器。这样下一级存储器中的数据都是最新的。

(2) 写回法。也称为拷回法。这种写策略只把数据写入 Cache 中相应的块，不写入下一级存储器。这样有些数据的最新版本是在 Cache 中。这些最新数据只有在相应的块被替换时，才被写回下一级存储器。

写回法和写直达法各有特色。写回法的优点是速度快，“写”操作能以 Cache 的速度进行。而且所有“写”只到达 Cache。对于同一单元的多次写来说，最后只需一次写回下一级存储器，因而减少了对存储器带宽的要求。这使得写回法对于多处理机很有吸引力。写直达法的优点是易于实现，而且下一级存储器中的数据总是最新的。后一个优点对于 I/O 和多处理机来说是重要的。

采用写直达法时，为了减少写停顿，常用方法是采用写缓冲器。CPU 一旦把数据写入该缓冲器，就可以继续执行，从而使下一级存储器的更新和 CPU 的执行重叠起来。

7. Cache 性能分析

虽然用不命中率来评价存储系统的性能非常方便，但它也容易产生一些误导。**平均访存时间**是一种更好的评测存储系统性能的指标。

$$\text{平均访存时间} = \text{命中时间} + \text{不命中率} \times \text{不命中开销}$$

平均访存时间的两个组成部分既可以用绝对时间，也可以用时钟周期数来衡量。

平均访存时间仍然是衡量性能的一个间接指标，尽管它比不命中率更好，但并不能代替程序执行时间。执行一个程序所需的 CPU 时间能更好地反映存储系统的性能。

$$\text{CPU 时间} = (\text{CPU 执行周期数} + \text{存储器停顿周期数}) \times \text{时钟周期时间} \tag{7.1}$$

在这个公式中，一般是把 Cache 命中所用的时钟周期数看成是 CPU 执行的时钟周期数的一部分。

$$\text{存储器停顿时钟周期数} = \text{访存次数} \times \text{不命中率} \times \text{不命中开销} \tag{7.2}$$

由于“读”和“写”的不命中率和不命中开销通常是不相等的，所以这只是一个近似公式。

把式(7.2)代入式(7.1)，得：

$$\text{CPU 时间} = (\text{CPU 执行周期数} + \text{访存次数} \times \text{不命中率} \times \text{不命中开销}) \times \text{时钟周期时间}$$

提取公因子“指令数”(IC)，得：

$$\begin{aligned}\text{CPU 时间} &= \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{访存次数}}{\text{指令数}} \times \text{不命中率} \times \text{不命中开销}\right) \times \text{时钟周期时间} \\ &= \text{IC} \times (\text{CPI}_{\text{execution}} + \text{每条指令的平均访存次数} \times \text{不命中率} \times \text{不命中开销}) \times \\ &\quad \text{时钟周期时间}\end{aligned}$$

Cache 的行为可能会对系统性能产生巨大的影响。而且，Cache 不命中对于一个 CPI 较小而时钟频率较高的 CPU 来说，影响是双重的。

(1) $CPI_{execution}$越低，固定周期数的 Cache 不命中开销的相对影响就越大。

(2) 在计算 CPI 时，不命中开销的单位是时钟周期数。因此，即使两台计算机的存储层次完全相同，时钟频率较高的 CPU 的不命中开销较大，其 CPI 中存储器停顿这部分也就较大。因此 Cache 对于低 CPI、高时钟频率的 CPU 来说更加重要。

尽可能地减少平均访存时间是一个合理的目标，而且在本章许多地方我们也是使用平均访存时间这个指标的，但是请记住，我们的最终目标是减少 CPU 的执行时间。对于一种 Cache 方案来说，如果它能减少平均访存时间，但会增加 CPU 时间，那么这种方案就不可取。

8. 改进 Cache 性能

根据平均访存时间公式：

$$平均访存时间=命中时间+不命中率\times不命中开销$$

可知，可以从以下 3 个方面改进 Cache 的性能：

(1) 降低不命中率；

(2) 减少不命中开销；

(3) 减少命中时间。

下面将介绍 17 种 Cache 优化技术，其中 8 种用于降低不命中率，5 种用于减少不命中开销，4 种用于减少命中时间。

7.2.3　降低 Cache 不命中率

本节介绍 8 种降低不命中率的方法。需要注意的是，许多降低不命中率的方法会增加命中时间或不命中开销。因此，在具体使用时，要综合考虑，保证降低不命中率确实能使整个系统速度提高。

1. 三种类型的不命中

按照产生不命中的原因不同，可以把不命中分为以下 3 类(简称为 3C)。

(1) 强制性不命中：当第一次访问一个块时，该块不在 Cache 中，需从下一级存储器中调入 Cache，这就是强制性不命中。这种不命中也称为冷启动不命中或首次访问不命中。

(2) 容量不命中：如果程序执行时所需的块不能全部调入 Cache 中，则当某些块被替换后，若又重新被访问，就会发生不命中。这种不命中称为容量不命中。

(3) 冲突不命中：在组相联或直接映像 Cache 中，若太多的块映像到同一组(块)中，则会出现该组中某个块被别的块替换然后又被重新访问的情况。这就是发生了冲突不命中。这种不命中也称为碰撞不命中或干扰不命中。

在 3C 中，冲突不命中似乎是最容易减少的，只要采用全相联，就不会发生冲突不命中。但是，用硬件实现全相联是很昂贵的，而且有可能会降低处理器的时钟频率，从而导致整体性能的下降。至于容量不命中，除了增大 Cache 以外，没有别的办法。

另一个减少3C的方法是增加块的大小,以减少强制性不命中。但在下面我们将看到,块大小增大可能会增加其他类型的不命中。

2. 增加Cache块大小

降低不命中率最简单的方法是增加块大小。但是,模拟结果表明:对于给定的Cache容量,当块大小从16字节开始增加时,不命中率开始是下降,但后来反而上升了。这是因为增加块大小会产生双重作用:

(1) 增强了空间局部性,减少了强制性不命中;

(2) 减少了Cache中块的数目,所以有可能会增加冲突不命中。

在块大小比较小的情况下,上述的第一种作用超过第二种作用,从而使不命中率下降。但等到块大小较大时,第二种作用超过第一种作用,就反而使不命中率上升了。Cache容量越大,使不命中率达到最低的块大小就越大。

3. 增加Cache的容量

降低Cache不命中率最直接的方法是增加Cache的容量。不过,这种方法不但会增加成本,而且还可能增加命中时间。这种方法在片外Cache中用得比较多。

4. 提高相联度

在相联度方面,有两条经验规则。①从实际应用的角度来看,在降低不命中率方面,8路组相联的作用已经和全相联一样有效。也就是说,采用相联度超过8的方案的实际意义不大。②2:1 Cache经验规则,即容量为N的直接映像Cache的不命中率和容量为$N/2$的两路组相联Cache的不命中率差不多相同。

一般来说,改进平均访存时间的某一方面是以损失另一方面为代价的。例如,增加块大小会增加不命中开销,而提高相联度则是以增加命中时间为代价。为了实现很高的处理器时钟频率,需要设计结构简单的Cache;但时钟频率越高,不命中开销就越大(所需的时钟周期数越多)。为减少不命中开销,又要求提高相联度。

5. 伪相联Cache

伪相联Cache又称为列相联Cache。它既能获得多路组相联Cache的低不命中率,又能保持直接映像Cache的命中速度。

当对伪相联Cache进行访问时,首先是按与直接映像相同的方式进行访问。如果命中,则从相应的块中取出所访问的数据,送给CPU,访问结束。这与直接映像Cache中的情况完全相同;但如果是不命中,就与直接映像Cache不同了,伪相联Cache会检查Cache另一个位置(块),看是否匹配。确定这个"另一块"的一种简单的方法是将索引字段的最高位取反,然后按照新索引去寻找"**伪相联组**"中的对应块。如果这一块的标识匹配,则称发生了"伪命中"。否则,就只好访问下一级存储器。

伪相联Cache具有一快一慢两种命中时间,它们分别对应于正常命中和**伪命中**的情况。为了使大多数命中都成为快速命中,要能够指出在同一组的两个块中访问哪个块才更可能是快速命中。一种简单的方法是:当出现伪命中时,交换两个块的内容,把最近刚访问过的

块放到第一位置(即按直接映像所对应的块)上。这是因为根据局部性原理,刚访问过的块很可能就是下一次要访问的块。

伪相联技术往往应用在离处理器比较远的Cache上,例如第二级Cache。

6. 硬件预取

指令和数据都可以在处理器提出访问请求之前进行预取。预取内容可以直接放入Cache,也可以放在一个访问速度比主存快的外部缓冲器中。指令预取通常由Cache之外的硬件完成。例如,Alpha AXP 21064微处理器在发生指令不命中时取两个块:被请求指令块和顺序的下一指令块。被请求指令块返回时放入Cache,而预取的指令块则放在缓冲器中。

7. 编译器控制的预取

这是另一种预取方法,是由编译器在程序中加入预取指令来实现预取,这些指令在数据被用到之前就将它们取到寄存器或Cache中。按照预取的处理方式不同,可把预取分为:

(1) 故障性预取:在预取时,若出现虚地址故障或违反保护权限,就会发生异常。

(2) 非故障性预取:当出现虚地址故障或违反保护权限时,不发生异常,而是会放弃预取,转变为空操作。

只有在预取数据的同时处理器还能继续执行的情况下,预取才有意义。这就要求Cache在等待预取数据返回的同时,还能继续提供指令和数据。这种灵活的Cache称为**非阻塞Cache**或**非锁定Cache**,将在后面详细讨论。

8. 编译优化

这种方法是通过对软件进行优化来降低不命中率。与其他降低Cache不命中率的方法相比,这种方法的特色是无须对硬件做任何改动。

1) 程序代码和数据重组

我们能很容易地重新组织程序而不影响程序的正确性。例如,把一个程序中的过程重新排序,就可能会减少冲突不命中,从而降低指令不命中率。另外,如果编译器知道一个分支指令很可能会成功转移,那么它就可以通过以下两步来改善空间局部性:①将转移目标处的基本块和紧跟着该分支指令后的基本块进行对调;②把该分支指令换为操作语义相反的分支指令。

与代码相比,数据对存储位置的限制更少,因此更便于调整顺序。对数据进行变换的目的是改善其空间局部性和时间局部性。编译优化技术包括数组合并、内外循环交换、循环融合、分块等。**数组合并**是将本来相互独立的多个数组合并成为一个复合数组,以提高访问它们的局部性。**循环融合**是将若干个独立的循环融合为单个的循环。这些循环访问同样的数组,对相同的数据做不同的运算。这样能使得读入Cache的数据在被替换出去之前,能得到反复的使用。

2) 内外循环交换

有些程序中含有嵌套循环,程序不是按照数据在存储器中存储的顺序进行访问。在这种情况下,只要简单地交换循环的嵌套关系,就能使程序按数据在存储器中存储的顺序进行

访问。这种技术是通过提高空间局部性来减少不命中次数。

3）分块

这种优化可能是Cache优化技术中最著名的一种，它是通过提高时间局部性来减少不命中。我们还是以对多个数组的访问为例，有些数组是按行访问，而有些则是按列访问。无论数组是按行优先还是按列优先存储，都不能解决问题，因为在每一次循环中既有按行访问也有按列访问。这种正交的访问意味着前面的变换方法，如内外循环交换，对此无能为力。

分块算法不是对数组的整行或整列进行访问，而是对子矩阵或块进行操作。其目的仍然是使一个Cache块在被替换之前最大限度地利用它。

9. “牺牲”Cache

这种方法是在Cache和其下一级存储器的数据通路之间增设一个全相联的小Cache，称为**“牺牲”Cache**。“牺牲”Cache中存放因冲突而被替换出去的那些块（即“牺牲者”）。每当发生不命中时，在访问下一级存储器之前，先检查“牺牲”Cache中是否含有所需的块。如果有，就将该块与Cache中某个块做交换，把所需的块从“牺牲”Cache调入Cache。

7.2.4 减少Cache不命中开销

1. 采用两级Cache

当一级Cache不能满足要求时，可以通过在原有Cache和存储器之间增设另一级Cache，构成两级Cache。这样，就可以把第一级Cache做得足够小，使其速度和快速CPU的时钟周期相匹配；同时，通过把第二级Cache做得足够大，使它能捕获更多本来需要到主存去的访问，从而降低实际不命中开销。

增加一级存储层次在概念上是直观和简单的。但其性能分析却变得复杂多了。用L1和L2分别表示第一级和第二级Cache，则原有的平均访存时间公式就变为：

$$平均访存时间 = 命中时间_{L1} + 不命中率_{L1} \times 不命中开销_{L1}$$

$$不命中开销_{L1} = 命中时间_{L2} + 不命中率_{L2} \times 不命中开销_{L2}$$

所以，

$$平均访存时间 = 命中时间_{L1} + 不命中率_{L1} \times (命中时间_{L2} + 不命中率_{L2} \times 不命中开销_{L2})$$

在这个公式里，第二级Cache的不命中率是以在第一级Cache中不命中而到达第二级Cache的访存次数为分母来计算的。为避免二义性，我们引入以下两个术语。

1）局部不命中率

对于某一级Cache来说：

局部不命中率 = 该级Cache的不命中次数 / 到达该级Cache的访存次数

对于第一级Cache来说，其局部不命中率就是上面的不命中率$_{L1}$，对于第二级Cache来说，就是上面的不命中率$_{L2}$。

2）全局不命中率

对于某一级Cache来说：

全局不命中率 = 该级Cache的不命中次数 /CPU发出的访存总次数

使用上面公式中的变量，第二级 Cache 的全局不命中率就是：

$$全局不命中率_{L2} = 不命中率_{L1} \times 不命中率_{L2}$$

全局不命中率是一个比局部不命中率更有用的衡量指标，它指出了在 CPU 发出的访存中，究竟有多大比例是穿过各级 Cache，最终到达存储器的。

采用两级 Cache 时，每条指令的平均访存停顿时间为：

$$\begin{aligned}每条指令的平均访存停顿时间 = &每条指令的平均不命中次数_{L1} \times \\ &命中时间_{L2} + 每条指令的平均不命中次数_{L2} \times \\ &不命中开销_{L2}\end{aligned}$$

对于第二级 Cache，有以下结论。

(1) 在第二级 Cache 比第一级 Cache 大得多的情况下，两级 Cache 的全局不命中率和容量与第二级 Cache 相同的单级 Cache 的不命中率非常接近。这时可以利用前面关于单级 Cache 的知识和结论来分析它们。

(2) 局部不命中率不是衡量第二级 Cache 的一个好指标，在评价第二级 Cache 时，应该用全局不命中率这个指标。

第一级 Cache 和第二级 Cache 之间的首要区别是：第一级 Cache 的速度会影响 CPU 的时钟频率，而第二级 Cache 的速度只影响第一级 Cache 的不命中开销。因此，在设计第二级 Cache 时可以有更多的考虑空间，许多不适合于第一级 Cache 的方案对于第二级 Cache 却可以使用。设计第二级 Cache 只有两个问题需要权衡，一个是它能否降低 CPI 中的平均访存时间部分？另一个是它的成本是多少？

第二级 Cache 的容量一般很大。大容量意味着第二级 Cache 可能实际上没有容量不命中，只剩下一些强制性不命中和冲突不命中。

2. 让读不命中优先于写

在写直达 Cache 中，每次写访问都要对主存进行写入。为了提高性能，一般都是设置一个大小适中的写缓冲器。不过，写缓冲器却导致存储器访问的复杂化，因为在读不命中时，所读单元的最新值有可能还在写缓冲器中，尚未写入主存。

解决这个问题常采用的办法，是在读不命中时检查写缓冲器的内容，看看有没有冲突。如果有冲突，就只好等待。否则就可继续处理读不命中。

在写回法 Cache 中，也可以利用写缓冲器来提高性能。假定读不命中将替换一个修改过的存储块。我们可以不像往常那样先把该块写回存储器，然后再从存储器调块，而是先把被替换的块临时复制到一个缓冲器中，然后从存储器调块，最后再把缓冲器中的内容写入存储器。这样 CPU 的读访问就能更快地完成。

3. 写缓冲合并

在写缓冲器不为空的情况下，需要把这次的写入地址与写缓冲器中已有的所有地址进行比较，看是否有匹配的项。如果有地址匹配而对应的位置又是空闲的，就把这次要写入的数据与该项合并。这就叫写**缓冲合并**。当然，如果写缓冲器满且没有能进行写合并的项，就只好等待。

4. 请求字处理技术

当从存储器向CPU调入一块时,块中往往只有一个字是CPU立即需要的,这个字称为请求字。当CPU所请求的字到达后,不等整个块都调入Cache,就可把该字发送给CPU并重启CPU继续执行。有以下两种具体的方案。

(1) 尽早重启动:在请求字没有到达时,CPU处于等待状态。一旦请求字到达,就立即发送给CPU,让等待的CPU尽早重启动,继续执行。

(2) 请求字优先:调块时,让存储器首先提供CPU所要的请求字。请求字一旦到达,就立即送给CPU,让CPU继续执行,同时从存储器调入该块的其余部分。请求字优先也称为关键字优先。

5. 非阻塞Cache技术

有些流水方式的机器采用计分牌或Tomasulo类控制方法,允许指令乱序执行,后面的指令可以跨越前面的指令先执行。CPU无须在Cache不命中时停顿。如果采用非阻塞Cache或非锁定Cache技术,就可以把CPU的性能提高得更多,因为这种Cache在不命中时仍允许CPU进行其他的访问(但只能是命中的访问)。这种"不命中下的命中"(hit under miss)的优化措施能减少实际的不命中开销。如果更进一步,让Cache允许多个不命中重叠,即支持"多重不命中下的命中"或"不命中下的不命中",则可进一步减少实际不命中开销。不过,这种方法只有在存储器能处理多个不命中的情况下才能带来好处。

7.2.5　减少命中时间

减少命中时间是设计Cache的重要工作之一,因为它直接影响到处理器时钟频率的高低。在当今的许多机器中,往往是Cache的访问时间限制了处理器系统时钟频率的提高,即使在把Cache访问时间分为几个时钟周期的机器中也是如此。因此,一定要设法减少命中时间。

1. 容量小、结构简单的Cache

为了有效地减少Cache的命中时间,可以采用容量小、结构简单的Cache。硬件越简单,速度就越快。而且应使Cache容量足够小,以便可以与处理器做在同一芯片上,避免因片外访问而增加时间开销。这一点是非常重要的。此外,还要保持Cache结构的简单性,例如采用直接映像Cache。直接映像Cache的主要优点是可以让标识检测和数据传送重叠进行,从而有效地减少命中时间。

2. 虚拟Cache

在采用虚拟存储器的计算机中,每次访存都必须进行虚实地址的转换,即将CPU发出的虚地址转换为物理地址,这一般是由存储管理部件MMU完成的。

1) 物理Cache

按照访问Cache的地址是物理地址还是虚拟地址,可把Cache分为物理Cache和虚拟

Cache。**物理 Cache** 是指使用物理地址进行访问的传统 Cache，其标识存储器中存放的是物理地址，进行地址检测也是用物理地址。当 CPU 要访问存储器时，必须先由 MMU 把虚拟地址转换为主存物理地址，然后再用物理地址去访问 Cache。这种 Cache 的性能很差。

2）虚拟 Cache

虚拟 Cache 是指可以直接用虚拟地址进行访问的 Cache，其标识存储器中存放的是虚拟地址，进行地址检测用的也是虚拟地址。当 CPU 要访问存储器时，把虚拟地址同时送给 Cache 和 MMU，Cache 根据该虚拟地址把 CPU 所需的数据或指令找出来。如果 Cache 不命中，就要用经过 MMU 转换得到的主存物理地址访问主存，读出相应的块，装入 Cache 中。虚拟 Cache 的优点是在命中时不需要地址转换，而且即使是不命中，地址转换和访问 Cache 也是并行进行的，其速度比物理 Cache 快很多。

然而，并非所有计算机都采用虚拟 Cache。其原因之一，是每当进行进程切换时需要清空 Cache。这是由于新进程的虚拟地址有可能与原进程的相同，但它们所指向的物理空间却是不同的。解决这个问题的一种办法是在地址标识中增加一个进程标识符字段（PID），这样多个进程的数据可以混合存放于 Cache 中，由 PID 指出 Cache 中的各块是属于哪个程序的。为了减少 PID 的位数，PID 经常是由操作系统指定。

虚拟 Cache 没有流行起来的另一个原因，是操作系统和用户程序对于同一个物理地址可能采用两种以上不同形式的虚拟地址来访问，这些地址称为同义或别名。它们可能会导致同一个数据在虚拟 Cache 中存在两个副本。而这是不允许的，否则就会发生错误。

用软件的办法来解决别名问题很容易，只要要求别名的某些地址位相同。这种限制被称为页着色。

3）虚拟索引-物理标识方法

这种方法既能得到虚拟 Cache 的好处，又能得到物理 Cache 的优点。它直接用虚地址中的页内位移作为访问 Cache 的索引，但标识却是物理地址。CPU 发出访存请求后，在进行虚→实地址转换的同时，可并行进行标识的读取。在完成地址转换之后，再把得到的物理地址与标识进行比较。

这种方法的局限性是直接映像 Cache 的容量不能超过页面的大小。为了既能实现大容量的 Cache，又能使索引位数比较少，以便能直接从虚拟地址的页内位移部分得到索引，可以采用提高相联度的办法。这一点可以从下面的公式中看出（其中，index 表示索引）：

$$\text{Cache 的容量} = 2^{index} \times \text{相联度} \times \text{块大小}$$

IBM 3033 的 Cache 采用了 16 路组相联。这样，尽管 IBM 系统结构限制了页的大小为 4KB，但 16 路组相联却使得可以用虚拟索引对 64KB(16×4KB)的 Cache 进行寻址。

3. Cache 访问流水化

这种技术把对第一级 Cache 的访问按流水方式组织，这样一来，就使得访问 Cache 需要多个时钟周期才可以完成。例如，Intel 的 Pentium 访问指令 Cache 需要一个时钟周期，Pentium Pro 到 Pentium Ⅲ需要两个时钟周期，而 Pentium 4 则需要 4 个时钟周期。这样处理的好处是可以提高时钟频率。实际上它并不能真正减少 Cache 的命中时间，但可以提高访问 Cache 的带宽。

4. 踪迹 Cache

开发指令级并行所遇到的一个挑战是：当要每个时钟周期流出超过4条指令时，要提供足够多条彼此互不相关的指令是很困难的。解决这个问题的一个方法，是采用踪迹Cache。普通的指令Cache都是存放静态指令序列的，与之不同，**踪迹 Cache** 中存放的是CPU所执行过的动态指令序列，其中包含由分支预测展开了的指令。该分支预测是否正确需要在取到该指令时进行确认。

踪迹Cache的地址映像机制比普通Cache的更复杂，但在另一方面，它能够提高指令Cache的空间利用率。对于普通Cache中的一个块来说，如果通过分支成功转到该块的某个位置开始执行，那么该块中处于该位置之前的部分就可能根本不会被用到。类似地，在往后面执行该块中的指令时，也可能因为遇到成功的分支而从该块中转移出去。这样，在该块中位于该分支指令之后的那些指令也可能是用不到的。这样会浪费不少空间。如果每5～10条指令就有一次跳转或成功分支，那么空间的浪费确实是个问题。踪迹Cache中则只存放上述从转入位置到转出位置之间的指令，从而避免了上述空间开销。

当然，踪迹Cache也有它的不足，就是相同的指令序列有可能被当作条件分支的不同选择而重复存放。

7.2.6 并行主存系统

并行主存系统是在一个访存周期内能并行访问到多个存储字的存储器，它能有效地提高存储器的带宽。

假设某存储器的访问周期是 T_M，字长为 W 位，则其带宽为：

$$B_M = \frac{W}{T_M}$$

在相同的器件条件下(T_M相同)，如果要提高主存的带宽，可以采用以下两种并行存储器结构：单体多字存储器，多体交叉存储器。

1. 单体多字存储器

如果一个存储器能够每个存储周期读出 m 个字，则其最大带宽提高到原来的 m 倍：

$$B_M = m \times \frac{W}{T_M}$$

单体多字并行存储器的优点是实现简单，缺点是访存效率不高，其原因包括以下4个方面。

(1) 单体多字并行存储器一次能读取 m 个指令字。如果这些指令字中有分支指令，而且分支成功，那么该分支指令之后的指令是无用的。

(2) 单体多字并行存储器一次取出的 m 个数据不一定都是有用的，而另一方面，当前执行指令所需要的多个操作数也不一定正好都存放在同一个长存储字中。由于数据存放的随机性比程序指令存放的随机性大，所以发生这种情况的概率较大。

(3) 在这种存储器中，必须凑齐了 m 个数之后才能一起写入存储器。如果只写个别字，

就必须先把相应的长存储字读出来，放到数据寄存器中，然后在地址码的控制下修改其中的一个字，最后再把长存储字写回存储器。

(4) 当要读出的数据字和要写入的数据字处于同一个“长存储字”内时，读和写的操作就无法在同一个存储周期内完成。

2. 多体交叉存储器

多体交叉存储器由多个单字存储体构成。每个体都有自己的地址寄存器以及地址译码和读/写驱动等电路。假设共有 m 个体，每一个体有 n 个存储单元。这 $n\times m$ 个单元可以看成是一个由存储单元构成的二维矩阵。但是，对于计算机使用者来说，存储器是按顺序线性编址的。如何在二维矩阵和线性地址之间建立对应关系？这就是对多体存储器如何进行编址的问题。

有两种编址方法：高位交叉编址，低位交叉编址。其中，只有低位交叉编址存储器才能有效地解决访问冲突问题。

1) 高位交叉编址

这种方式相当于对存储单元矩阵按列优先的方式进行编址。同一个体中的高 $\log_2 m$ 位都是相同的，这就是体号。考虑处于第 i 行第 j 列的单元，即体号为 j、体内地址为 i 的单元，其线性地址可按下式求得：

$$A=j\times n+i\quad (其中, j=0,1,2,\cdots,m-1;\ i=0,1,2,\cdots,n-1)$$

反过来，如果已经知道一个单元的线性地址为 A，则其体号 j 和体内地址 i 可按以下公式求得：

$$j=\left\lfloor\frac{A}{n}\right\rfloor$$

$$i=A \bmod n$$

如果把 A 表示为二进制数，则其高 $\log_2 m$ 位就是体号，而剩下的部分就是体内地址。

2) 低位交叉编址

这种方式相当于对存储单元矩阵按行优先进行编址。即先给第 0 行的各单元按从左到右的顺序依次赋予地址，然后再给第 1 行的各单元按顺序依次赋予地址，…，最后给最后一行的各单元按顺序依次赋予地址。这样，同一个体中的低 $\log_2 m$ 位都是相同的，这就是体号。

考虑处于第 i 行第 j 列的单元，即体号为 j、体内地址为 i 的单元，其线性地址可按下式求得：

$$A=i\times m+j\quad (其中, i=0,1,2,\cdots,n-1;\ j=0,1,2,\cdots,m-1)$$

反过来，如果已经知道一个单元的线性地址为 A，则其体号 j 和体内地址 i 可按以下公式求得：

$$j=\left\lfloor\frac{A}{m}\right\rfloor$$

$$i=A \bmod m$$

如果把 A 表示为二进制数，则其低 $\log_2 m$ 位就是体号，而剩下的部分就是体内地址。

为了提高主存的带宽，需要多个或所有存储体能并行工作。由于程序执行过程中，

CPU所访问的指令和数据的地址是按顺序连续的，所以必须采用低位交叉访问的存储器，并在每一个存储周期内，分时启动 m 个存储体。

虽然在理想情况下，这种存储器的带宽最高能提高到原来的 m 倍。但是，由于存在访问冲突，实际加速比小于 m。单纯靠增大 m 来提高并行主存系统的带宽是有限的，而且性能价格比还会随 m 的增大而下降。

3. 避免存储体冲突

在许多情况下，都要求存储系统能支持多个独立的访存请求。这时存储器系统的性能将取决于这些独立的访存请求发生体冲突的频度的高低。所谓体冲突，是指两个访问请求要访问同一个存储体。在传统的多体交叉结构中，顺序访问被处理得很好，不会发生体冲突。地址相差奇数值的访存也是如此。问题是当地址相差偶数值时，冲突的频度就增加了。解决这个问题的一种方法，是采用很多个体去减少体冲突的次数。这种方法只有在较大规模的机器中才采用。

体冲突问题既可以用软件方法也可以用硬件方法来解决。编译器可以通过循环交换优化来避免对同一个体的访问。更简单的一种方法是让程序员或编译器来扩展数组的大小，使之不是2的幂，从而强制使所访问的地址落在不同的体内。

减少体冲突的一种硬件解决方案是使体数为素数。采用素数看起来似乎会需要更多的硬件来完成复杂的计算，会延长每次访存的时间。幸运的是，有几种硬件方法能快速地进行上述计算。尤其是当存储体数为素数、且为2的幂减1时，可以用下面的计算来代替除法运算：

$$\text{体内地址 } j = A \bmod n$$

由于一个存储体中包含的字数 n 一般是2的幂，所以可以用位选择方法来实现上述计算。

7.2.7　虚拟存储器

1. 基本概念

虚拟存储器是“主存-辅存”层次进一步发展的结果。它由一个主存储器和一个容量很大的辅助存储器(通常是硬盘)组成，在系统软件和辅助硬件的管理下，就像一个单一的、可直接访问的大容量主存储器。应用程序员可以用机器指令的地址码对整个程序统一编址，就如同应用程序员具有对应于这个地址码宽度的存储空间(称为程序空间)一样。

2. 快速地址转换技术

页表一般都很大，存放在主存中。这样，每次访存都要引起对主存的两次访问：第一次是访问页表，以获得所要访问数据的物理地址；第二次才是访问数据本身。显然，这使得对存储器的访问速度至少下降了一倍，是无法实用的。一般采用TLB来解决这个问题。

TLB(Translation Look-aside Buffer)是一个专用的高速缓冲器，用于存放近期经常使用的页表项(Page Table Entry，PTE)。大多数访存都可以通过TLB快速地完成虚→实地址转换。只有偶尔在TLB不命中时，才需要去访问主存中的页表。

TLB 中的项与 Cache 中的项类似，也是由两部分构成：标识和数据。标识中存放的是虚地址的一部分，而数据部分中存放的则是物理页帧号、有效位、存储保护信息、使用位、修改位等。为了使 TLB 中的内容与页表保持一致，当修改页表中的某一项时，操作系统必须保证 TLB 中没有该页表项的副本。这可以通过作废 TLB 中的页表项来实现。

地址转换很容易处在确定处理器时钟周期的关键路径上，所以，一般 TLB 比 Cache 的标识存储器更小，而且更快，这样才能保证 TLB 的读出操作不会使 Cache 的命中时间延长。

习　题

1. 概念题

【题 7.1】　解释下列名词

多级存储层次	命中时间	不命中率	不命中开销
全相联映像	直接映像	组相联映像	替换算法
LRU	写直达法	写回法	按写分配法
不按写分配法	强制性不命中	容量不命中	冲突不命中
2∶1Cache 经验规则	相联度	Victim Cache	故障性预取
非故障性预取	非阻塞 Cache	尽早重启动	请求字优先
多级包容性	虚拟 Cache	并行主存系统	多体交叉存储器
存储体冲突	TLB		

2. 选择题

【题 7.2】　程序员编写程序时，使用的访存地址是(　　)。

A. 主存地址　　B. 逻辑地址　　C. 物理地址　　D. 有效地址

【题 7.3】　虚拟存储器主要是为了(　　)。

A. 扩大存储系统的容量

B. 提高存储系统的速度

C. 扩大存储系统的容量和提高存储系统的速度

D. 便于程序的访存操作

【题 7.4】　与全相联映像相比，组相联映像的优点是(　　)。

A. 目录表小　　B. 块冲突概率低　　C. 命中率高　　D. 主存利用率高

【题 7.5】　按 Cache 地址映像的块冲突概率从高到低的顺序是(　　)。

A. 全相联映像、直接映像、组相联映像

B. 组相联映像、直接映像、全相联映像

C. 直接映像、组相联映像、全相联映像

D. 全相联映像、组相联映像、直接映像

【题 7.6】　对于采用组相联映像、LRU 替换算法的 Cache 存储器来说，不影响 Cache 命中率的是(　　)。

A. 增加 Cache 中的块数　　　　B. 增大组的大小

C. 增大主存容量　　　　D. 增大块的大小

【题 7.7】 下列说法不正确的是(　　)。

A. 单体多字存储器能提高存储器频宽

B. 多体存储器低位交叉编址能提高存储器频宽

C. 多体存储器高位交叉编址便于扩大存储器容量

D. 多体存储器高位交叉编址能提高存储器频宽

3. 填空题

【题 7.8】 存储层次的性能参数有______、______、______和______4个。

【题 7.9】 存储器层次结构设计技术的基本依据是程序的______原理，它包括______和______两方面。

【题 7.10】 "主存-辅存"层次的目的是为了弥补主存______的不足；"Cache-主存"层次的目的是为了弥补主存______的不足。

【题 7.11】 设有一个"Cache-主存"层次，Cache 为 8 块，主存为 16 块；试分别对于以下两种情况，计算访存块地址为 6 时的索引(index)。①组相联，每组两块；索引为______；②直接映像；索引为______。

【题 7.12】 存储层次要解决的 4 个问题是______、______、______和______。

【题 7.13】 在"Cache-主存"层次中，CPU 的访存地址被分割为______和______两部分。

【题 7.14】 Cache 存储器采用组相联映像是指组间______映像，组内各块之间是______映像。

【题 7.15】 当组相联的路数 n 变为 1 时，组相联就变成了______，当组数 G 变为 1 时，组相联就变成了______。

【题 7.16】 在 Cache 存储器中，用比较对法实现 LRU 替换算法时，当 Cache 的块数为 8 时，需要的触发器个数为______。

【题 7.17】 用堆栈按 LRU 替换算法对访存虚页地址流处理完后，从栈顶到栈底存放的虚页号是各虚页______的顺序。

【题 7.18】 在"Cache-主存"层次中，主存的更新算法有______和______两种。

【题 7.19】 在"Cache-主存"层次中，写回法 Cache 一般采用______更新主存，写直达法 Cache 一般采用______更新主存。

【题 7.20】 随机法中______选择被替换的块。先进先出法中选择______作为被替换的块。最近最少使用法中选择______作为被替换的块。

【题 7.21】 按照产生不命中的原因不同，可以把不命中分为______、______和______3类。

【题 7.22】 相联度越高，______就越少；______不受 Cache 容量的影响，但______却随着容量的增加而减少；______和______不受相联度的影响。

【题 7.23】 对于给定的 Cache 容量，当块大小增加时，不命中率开始是______，后来反而______。Cache 容量越大，使不命中率达到最低的块大小就______。

【题 7.24】 增加块大小的方法在降低不命中率的同时会增加________，而提高相联度会增加________。

【题 7.25】 伪相联既能获得________ Cache 的低不命中率，又能保持________ Cache 的命中速度。

【题 7.26】 操作系统和用户程序对于同一个物理地址可能采用两种以上不同形式的虚拟地址来访问，这些地址称为________或________。

【题 7.27】 在相同的器件条件下，如果要提高主存的带宽，可以采用________和________两种并行存储器结构。

【题 7.28】 虚拟存储器采用________映像规则，替换算法采用________算法，写策略采用________。

4. 问答题

【题 7.29】 单级存储器的主要矛盾是什么？通常采取什么方法来解决？

【题 7.30】 简述“Cache-主存”层次与“主存-辅存”层次的区别。

【题 7.31】 地址映像方法有哪几种？它们各有什么优缺点？

【题 7.32】 替换算法有哪几种？它们各有什么优缺点？

【题 7.33】 写策略主要有哪两种？它们各有什么优点？

【题 7.34】 画出 4 路组相联并行标识比较的情况。

【题 7.35】 简述伪相联的基本思想。

【题 7.36】 增加块大小导致 Cache 不命中率先下降后上升的原因是什么？

【题 7.37】 降低 Cache 不命中率有哪几种方法？简述其基本思想。

【题 7.38】 简述减小 Cache 不命中开销的几种方法。

【题 7.39】 请求字处理技术有哪两种具体的实现方法？

【题 7.40】 简述采用二级 Cache 的基本思想。

【题 7.41】 通过编译器对程序优化来改进 Cache 性能的方法有哪几种？简述其基本思想。

【题 7.42】 组相联 Cache 的不命中率比相同容量直接映像 Cache 的不命中率低。由此能否得出结论：采用组相联一定能带来性能上的提高？为什么？

【题 7.43】 写出三级 Cache 的平均访问时间的公式。

【题 7.44】 简述采用容量小且结构简单的 Cache 所带来的好处。

【题 7.45】 简述“虚拟索引＋物理标识” Cache 的基本思想。它有什么优缺点？

【题 7.46】 在设计 Cache 时并非都采用虚拟 Cache，为什么？

【题 7.47】 单体多字并行存储器的访存效率不高，其原因是什么？

5. 应用题

【题 7.48】 VAX-11/780 在 Cache 命中时的指令平均执行时间是 9.5 时钟周期，Cache 不命中时间是 5 个时钟周期。假设不命中率是 12%，每条指令平均访存 2.5 次。试计算考虑了 Cache 不命中时的指令平均执行时间。它比 Cache 命中时的指令平均执行时间延长了百分之几？

【题 7.49】 我们考虑某一个机器。假设 Cache 读不命中开销为 25 个时钟周期，写不命中开销为 70 个时钟周期，当不考虑存储器停顿时，所有指令的执行时间都是 2.0 个时钟周期，Cache 的读不命中率和写不命中率均为 4%，平均每条指令读存储器 0.8 次，写存储器 0.5 次。试分析考虑 Cache 的不命中后，Cache 对性能的影响。

【题 7.50】 假设对指令 Cache 的访问占全部访问的 75%；而对数据 Cache 的访问占全部访问的 25%。Cache 的命中时间为 1 个时钟周期，不命中开销为 50 个时钟周期，在统一 Cache 中一次 load 或 store 操作访问 Cache 的命中时间都要增加一个时钟周期，32KB 的指令 Cache 的不命中率为 0.39%，32KB 的数据 Cache 的不命中率为 4.82%，64KB 的统一 Cache 的不命中率为 1.35%。又假设采用写直达策略，且有一个写缓冲器，并且忽略写缓冲器引起的等待。试问指令 Cache 和数据 Cache 容量均为 32KB 的分离 Cache 和容量为 64KB 的统一 Cache 相比，哪种 Cache 的不命中率更低？两种情况下平均访存时间各是多少？

【题 7.51】 假定存储系统在延迟 30 个时钟周期后，每两个时钟周期能送出 16 个字节。即：经过 32 个时钟周期，它可提供 16 个字节；经过 34 个时钟周期，可提供 32 个字节；以此类推。命中时间与块大小无关，为 1 个时钟周期，分别计算下列各种容量的 Cache 的平均访存时间。①块大小为 32 个字节，Cache 容量为 1KB，不命中率为 13.34%；②块大小为 32 个字节，Cache 容量为 4KB，不命中率为 7.21%；③块大小为 64 个字节，Cache 容量为 16KB，不命中率为 2.64%；④块大小为 128 个字节，Cache 容量为 16KB，不命中率为 2.77%。

【题 7.52】 Alpha AXP 21064 中，16KB 指令 Cache 的不命中率为 0.64 %，命中时间为 1 个时钟周期，不命中开销为 60 个时钟周期。假设采用指令预取技术后，预取命中率为 30 %。当指令不在指令 Cache 里，而在预取缓冲器中找到时，需要多花一个时钟周期。其实际不命中率是多少？

【题 7.53】 假设在 3000 次访存中，第一级 Cache 不命中 110 次，第二级 Cache 不命中 55 次。试问：在这种情况下，该 Cache 系统的局部不命中率和全局不命中率各是多少？

【题 7.54】 在三级 Cache 中，第一级 Cache、第二级 Cache 和第三级 Cache 的局部不命中率分别为 4%、30%和 50%。它们的全局不命中率各是多少？

【题 7.55】 给定以下的假设，试计算直接映像 Cache 和两路组相联 Cache 的平均访问时间以及 CPU 的性能。由计算结果能得出什么结论？

(1) 理想 Cache 情况下的 CPI 为 2.0，时钟周期为 2ns，平均每条指令访存 1.2 次；

(2) 两者 Cache 容量均为 64KB，块大小都是 32B；

(3) 组相联 Cache 中的多路选择器使 CPU 的时钟周期增加了 10%；

(4) 这两种 Cache 的不命中开销都是 80ns；

(5) 命中时间为 1 个时钟周期；

(6) 64KB 直接映像 Cache 的不命中率为 1.4%，64KB 两路组相联 Cache 的不命中率为 1.0%。

【题 7.56】 假设一台计算机具有以下特性：

(1) 95%的访存在 Cache 中命中；

(2) 块大小为两个字，且不命中时整个块被调入；

(3) CPU 发出访存请求的速率为 10^9 字/秒；

(4) 25%的访存为写访问；

(5) 存储器的最大流量为 10^9 字/秒(包括读和写)；

(6) 主存每次只能读或写一个字；

(7) 在任何时候，Cache 中有 30%的块被修改过；

(8) 写不命中时，Cache 采用按写分配法。

现欲给该计算机增添一台外设，为此首先想知道主存的频带已用了多少。试对于以下两种情况计算主存频带的平均使用比例。

(1) 写直达 Cache；

(2) 写回法 Cache。

【题 7.57】 在伪相联中，假设在直接映像位置没有发现匹配，而在另一个位置才找到数据(伪命中)时，不对这两个位置的数据进行交换。这时只需要 1 个额外的周期。假设不命中开销为 50 个时钟周期，2KB 直接映像 Cache 的不命中率为 9.8%，2KB 2 路组相联的不命中率为 7.6%；128KB 直接映像 Cache 的不命中率为 1.0%，128KB 2 路组相联的不命中率为 0.7%。

(1) 推导出平均访存时间的公式。

(2) 利用(1)中得到的公式，对于 2KB Cache 和 128KB Cache，计算伪相联的平均访存时间。

【题 7.58】 假设采用理想存储器系统时的基本 CPI 是 1.5，主存延迟是 40 个时钟周期；传输速率为 4 字节/时钟周期，且 Cache 中 50%的块是修改过的。每个块中有 32 字节，20%的指令是数据传送指令。并假设没有写缓存，在 TLB 不命中的情况下需要 20 时钟周期，TLB 不会降低 Cache 命中率。CPU 产生指令地址或 Cache 不命中时产生的地址有 0.2%没有在 TLB 中找到。

(1) 在理想 TLB 情况下，计算均采用写回法 16KB 直接映像统一 Cache、16KB 两路组相联统一 Cache 和 32KB 直接映像统一 Cache 机器的实际 CPI；

(2) 在实际 TLB 情况下，用(1)的结果，计算均采用写回法 16KB 直接映像统一 Cache、16KB 两路组相联统一 Cache 和 32KB 直接映像统一 Cache 机器的实际 CPI。

其中，假设 16KB 直接映像统一 Cache、6KB 两路组相联统一 Cache 和 32KB 直接映像统一 Cache 的不命中率分别为 2.9%、2.2%和 2.0%；25%的访存为写访问。

【题 7.59】 某个程序共访问存储器 1 000 000 次，该程序在某个系统中运行，系统中 Cache 的不命中率为 7%，其中，强制性不命中和容量不命中各占 25%，冲突不命中占 50%。问：

(1) 当允许对该 Cache 所做的唯一改变是提高相联度时，此时期望能够消除的最大不命中次数是多少？

(2) 当允许同时提高 Cache 的容量大小和相联度时，此时期望能够消除的最大不命中次数是多少？

【题 7.60】 设主存每个分体的存储周期为 2μs，存储字长为 4B，采用 m 个分体低位交叉编址。由于各种原因，主存实际带宽只能达到最大带宽的 0.6 倍，现要求主存实际带宽为 4MB/s，问主存分体数应取多少？

【题7.61】 设主存由8个存储体按低位交叉编址方式组成，主存容量1M字，Cache容量为4K字，要求一个主存周期从主存取得一个块。采用全相联地址映像，用相联目录表实现地址变换。求出相联目录表的行数、比较位数、宽度和总位数。

【题7.62】 程序存放在模32单字交叉存储器中，设访存申请队列的转移概率λ为25%，求每个存储周期能访问到的平均字数。当模数为16时呢？由此可得出什么结论？

题　　解

1. 概念题

【题7.1】 解释下列名词

多级存储层次——采用不同技术实现的存储器构成的一个存储系统。处在离CPU不同距离的层次上，各存储器之间一般满足包容关系，即任何一层存储器中的内容都是其下一层(离CPU更远的一层)存储器中内容的子集。目标是达到离CPU最近的存储器的速度，最远的存储器的容量。

命中时间——访问Cache命中时所用的时间。

不命中率——CPU访存时，在一级存储器中找不到所需信息的概率。

不命中开销——CPU向二级存储器发出访问请求到把这个数据调入一级存储器所需的时间。

全相联映像——主存中的任一块可以被放置到Cache中任意一个地方。

直接映像——主存中的每一块只能被放置到Cache中唯一的一个地方。

组相联映像——将Cache分成若干个组，每组由若干块构成。主存中的每一块可以放置到Cache中唯一的一组中任何一个地方。

替换算法——由于主存中的块比Cache中的块多，所以当要从主存中调一个块到Cache中时，会出现该块所映像到的一组(或一个)Cache块已全部被占用的情况。这时，需被迫腾出其中的某一块，以接纳新调入的块。

LRU——选择最近最少被访问的块作为被替换的块。实际实现都是选择最久没有被访问的块作为被替换的块。

写直达法——在执行写操作时，不仅把信息写入Cache中相应的块，而且也写入下一级存储器中相应的块。

写回法——只把信息写入Cache中相应块，该块只有在被替换时，才被写回主存。

按写分配法——写不命中时，先把所写单元所在的块调入Cache，然后再进行写入。

不按写分配法——写不命中时，直接写入下一级存储器中，而不把相应的块调入Cache。

强制性不命中——当第一次访问一个块时，该块不在Cache中，需要从下一级存储器中调入Cache，这就是强制性不命中。

容量不命中——如果程序在执行时，所需要的块不能全部调入Cache中，则当某些块被替换后又重新被访问，就会产生不命中，这种不命中就称作容量不命中。

冲突不命中——在组相联或直接映像 Cache 中，若太多的块映像到同一组(块)中，则会出现该组中某个块被别的块替换(即使别的组或块有空闲位置)，然后又被重新访问的情况。

2∶1Cache 经验规则——大小为 N 的直接映像 Cache 的不命中率约等于大小为 $N/2$ 的两路组相联 Cache 的不命中率。

相联度——在组相联中，每组所包含的块数。

Victim Cache——位于 Cache 和存储器之间的又一级 Cache，容量小，采用全相联策略。用于存放由于不命中而被丢弃(替换)的那些块。每当不命中发生时，在访问下一级存储器之前，先检查 Victim Cache 中是否含有所需块。

故障性预取——在预取时，若出现虚地址故障或违反保护权限，就会发生异常。

非故障性预取——在预取时，若出现虚地址故障或违反保护权限，不发生异常。

非阻塞 Cache——Cache 在等待预取数据返回时，还能继续提供指令和数据。

尽早重启动——在请求字没有到达时，CPU 处于等待状态。一旦请求字到达，就立即发送给 CPU，让等待的 CPU 尽早重启动，继续执行。

请求字优先——调块时，首先向存储器请求 CPU 所要的请求字。请求字一旦到达，就立即送往 CPU，让 CPU 继续执行，同时从存储器调入该块的其余部分。

多级包容性——一级存储器(Cache)中的数据总位于下一级存储器中。

虚拟 Cache——地址使用虚地址的 Cache。

并行主存系统——在一个访存周期内能并行访问到多个存储字的存储器，它能有效地提高存储器的带宽。

多体交叉存储器——具有多个存储体，各体之间按字交叉的存储器。

存储体冲突——多个请求要访问同一个体。

TLB——一个专用高速存储器，用于存放近期经常使用的页表项，其内容是页表部分内容的一个副本。

2. 选择题

【题 7.2】　答：B

【题 7.3】　答：A

【题 7.4】　答：A

【题 7.5】　答：C

【题 7.6】　答：C

【题 7.7】　答：D

3. 填空题

【题 7.8】　答：存储容量、平均每位价格、命中率、平均访问时间

【题 7.9】　答：局部性、空间局部性、时间局部性

【题 7.10】　答：容量、速度

【题 7.11】　答：01、0

【题 7.12】　答：映像规则、查找方法、替换算法、写策略

【题 7.13】　答：块地址、块内位移

【题7.14】　答：直接、全相联

【题7.15】　答：直接映像、全相联

【题7.16】　答：28

【题7.17】　答：按时间从最近到过去最久被访问

【题7.18】　答：按写分配法、不按写分配法

【题7.19】　答：按写分配法、不按写分配法

【题7.20】　答：随机地、最早调入的块、近期最少被访问的块

【题7.21】　答：强制性不命中、容量不命中、冲突不命中

【题7.22】　答：冲突不命中、强制性不命中、容量不命中、强制性不命中、容量不命中

【题7.23】　答：下降、上升、越大

【题7.24】　答：不命中开销、命中时间

【题7.25】　答：多路组相联、直接映像

【题7.26】　答：同义、别名

【题7.27】　答：单体多字存储器、多体交叉存储器

【题7.28】　答：全相联、最近最少使用LRU、写回策略

4. 问答题

【题7.29】　答：①速度越快，每位价格就越高；②容量越大，每位价格就越低；③容量越大，速度越慢。采取多级存储层次方法来解决。

【题7.30】　答："Cache-主存"层次与"主存-辅存"层次的区别见表7.2。

表7.2　"Cache-主存"层次与"主存-辅存"层次的区别

比较项目＼存储层次	"Cache-主存"层次	"主存-辅存"层次
目的	为了弥补主存速度的不足	为了弥补主存容量的不足
存储管理的实现	全部由专用硬件实现	主要由软件实现
访问速度的比值(第一级比第二级)	几比一	几万比一
典型的块(页)大小	几十个字节	几百到几千个字节
CPU对第二级的访问方式	可直接访问	均通过第一级
不命中时CPU是否切换	不切换	切换到其他进程

【题7.31】　答：①全相联映像。实现查找的机制复杂，代价高，速度慢。Cache空间的利用率较高，块冲突概率较低，因而Cache的不命中率也低。②直接映像。实现查找的机制简单，速度快。Cache空间的利用率较低，块冲突概率较高，因而Cache的不命中率也高。③组相联映像。组相联是直接映像和全相联的一种折中。

【题7.32】　答：①随机法。简单、易于用硬件实现，但这种方法没有考虑Cache块过去被使用的情况，反映不了程序的局部性，所以其不命中率比LRU的高。②先进先出法。容易实现。它虽然利用了同一组中各块进入Cache的顺序这一"历史"信息，但还是不能正确地反映程序的局部性。③最近最少使用法LRU。不命中率最低。但是LRU比较复杂，硬

件实现比较困难。

【题 7.33】 答：①写直达法。易于实现，而且下一级存储器中的数据总是最新的。②写回法。速度快，"写"操作能以 Cache 存储器的速度进行。而且对于同一单元的多个写最后只需一次写回下一级存储器，有些"写"只到达 Cache，不到达主存，因而所使用的存储器频带较低。

【题 7.34】 答：4 路组相联并行标识比较的情况如图 7.4 所示。

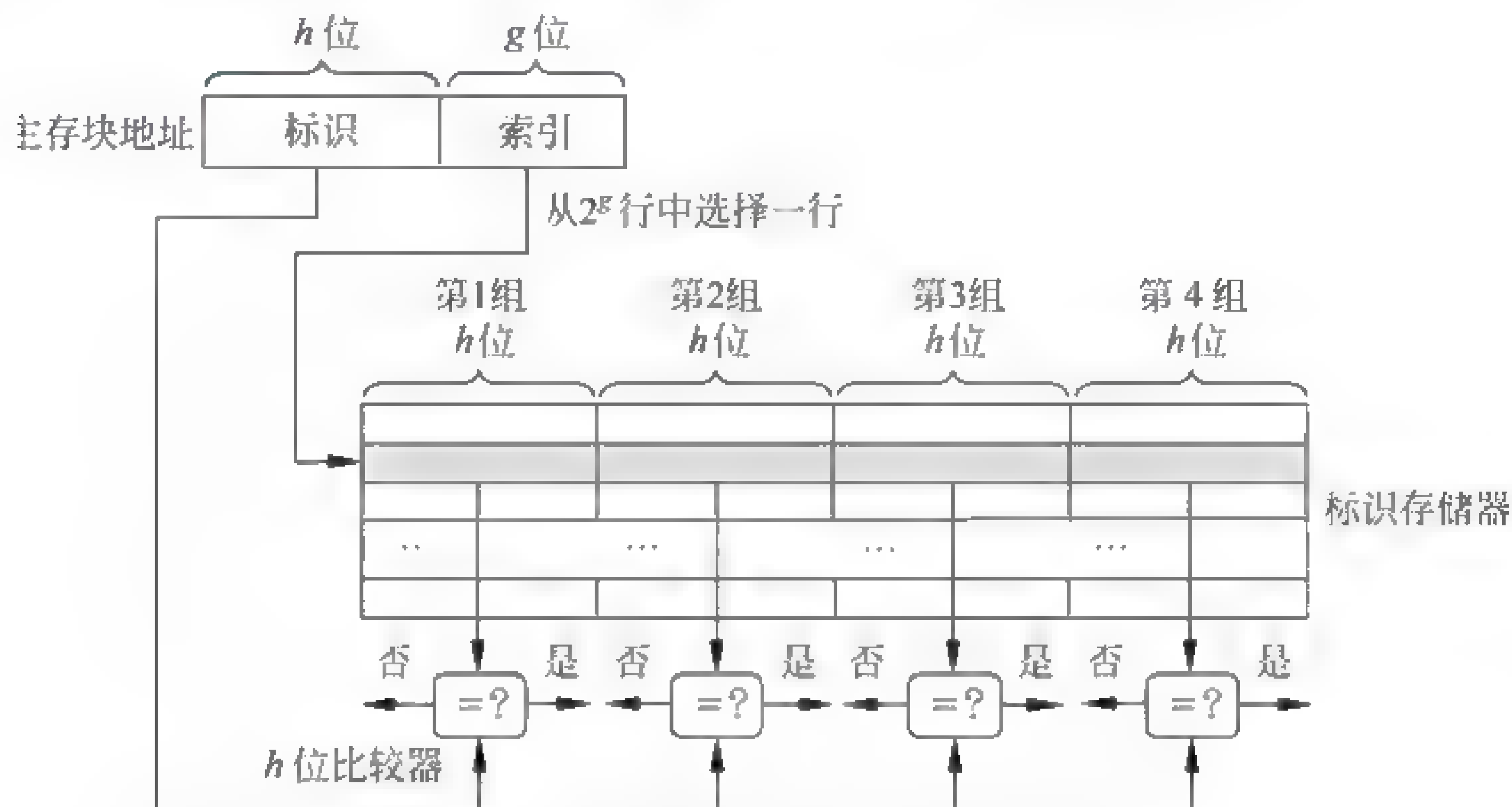


图 7.4　4 路组相联并行标识比较的情况

【题 7.35】 答：采用这种方法时，在命中情况下，访问 Cache 的过程和直接映像 Cache 中的情况相同；而发生不命中时，在访问下一级存储器之前，会先检查 Cache 另一个位置（块），看是否匹配。确定这个另一块的一种简单的方法是将索引字段的最高位取反，然后按照新索引去寻找伪相联组中的对应块。如果这一块的标识匹配，则称发生了伪命中。否则，就只好访问下一级存储器。

【题 7.36】 答：(1) 一方面它减少了强制性不命中，因为局部性原理有两方面的含义：时间局部性和空间局部性，增加块大小利用了时间局部性。

(2) 另一方面，由于增加块大小会减少 Cache 中块的数目，所以有可能会增加冲突不命中。在 Cache 容量较小时，甚至还会增加容量不命中。刚开始增加块大小时，由于块大小还不是很大，上述的第一种作用超过第二种作用，从而使不命中率下降。但等到块大小较大时，第二种作用超过第一种作用，使不命中率上升。

【题 7.37】 答：常用的降低 Cache 不命中率的方法有下面几种。

(1) 增加 Cache 块大小。增加块大小利用了程序的空间局部性。

(2) 增加 Cache 的容量。

(3) 提高相联度，降低冲突不命中。

(4) 伪相联 Cache，降低冲突不命中。当对伪相联 Cache 进行访问时，首先是按与直接映像相同的方式进行访问。如果命中，则从相应的块中取出所访问的数据，送给 CPU，访问结束。如果不命中，就将索引字段的最高位取反，然后按照新索引去寻找"伪相联组"中的对应块。如果这一块的标识匹配，则称发生了"伪命中"。否则，就访问下一级存储器。

(5) 硬件预取技术。指令和数据都可以在处理器提出访问请求前进行预取。

(6) 由编译器控制的预取。硬件预取的替代方法,在编译时加入预取的指令,在数据被用到之前发出预取请求。

(7) 编译器优化。通过对软件的优化来降低不命中率。

(8) "牺牲"Cache。在Cache和其下一级存储器的数据通路之间增设一个全相联的小Cache,存放因冲突而被替换出去的那些块。每当发生不命中时,在访问下一级存储器之前,先检查"牺牲"Cache中是否含有所需的块。如果有,就将该块与Cache中某个块做交换,把所需的块从"牺牲"Cache调入Cache。

【题7.38】 答:(1) 让读不命中优先于写。

(2) 写缓冲合并。

(3) 请求字处理技术。

(4) 非阻塞Cache或非锁定Cache技术。

(5) 采用二级Cache。

【题7.39】 答:(1) 尽早重启动:在请求字没有到达时,CPU处于等待状态。一旦请求字到达,就立即发送给CPU,让等待的CPU尽早重启动,继续执行。

(2) 请求字优先:调块时,首先向存储器请求CPU所要的请求字。请求字一旦到达,就立刻送往CPU,让CPU继续执行,同时从存储器调入该块的其余部分。请求字优先也称为回绕读取或关键字优先。

【题7.40】 答:通过在原有Cache和存储器之间增加另一级Cache,构成两级Cache,我们可以把第一级Cache做得足够小,使其速度和快速CPU的时钟周期相匹配,而把第二级Cache做得足够大,使它能捕获更多本来需要到主存去的访问,从而降低实际不命中开销。

【题7.41】 答:①数组合并。通过提高空间局部性来减少不命中次数。有些程序同时用相同的索引来访问若干个数组的同一维,这些访问可能会相互干扰,导致冲突不命中,可以将这些相互独立的数组合并成一个复合数组,使得一个Cache块中能包含全部所需元素。②内外循环交换。循环嵌套时,程序没有按数据在存储器中的顺序访问。只要简单地交换内外循环,就能使程序按数据在存储器中的存储顺序进行访问。③循环融合。有些程序含有几部分独立的程序段,它们用相同的循环访问同样的数组,对相同的数据做不同的运算。通过将它们融合成一个单一循环,能使读入Cache的数据被替换出去之前得到反复的使用。④分块。通过改进时间局部性来减少不命中。分块不是对数组的整行或整列进行访问,而是对子矩阵或块进行操作。

【题7.42】 答:不一定。因为组相联命中率的提高是以增加命中时间为代价的,组相联需要增加多路选择开关。

【题7.43】 答:平均访存时间=命中时间+不命中率×不命中开销

只有第I层的不命中时才会访问第$I+1$层。

设三级Cache的命中率分别为H_{L1}、H_{L2}、H_{L3},不命中率分别为M_{L1}、M_{L2}、M_{L3},第三级Cache的不命中开销为P_{L3}。

$$\text{平均访问时间 } T_A = H_{L1} + M_{L1}\{H_{L2} + M_{L2}(H_{L3} + M_{L3} \times P_{L3})\}$$

【题7.44】 答:①可以有效地提高Cache的访问速度。因为硬件越简单,速度就越快。小容量Cache可以实现快速标识检测,对减少命中时间有益。②Cache足够小,可以与处理

器做在同一芯片上，以避免因芯片外访问而增加时间开销。③保持 Cache 结构简单可采用直接映像 Cache。直接映像 Cache 的主要优点是可以让标识检测和数据传送重叠进行，这样可以有效地减少命中时间。

【题 7.45】 答：直接用虚地址中的页内位移(页内位移在虚→实地址的变换中保持不变)作为访问 Cache 的索引，但标识却是物理地址。CPU 发出访存请求后，在进行虚→实地址变换的同时，可并行进行标识的读取。在完成地址变换之后，再把得到的物理地址与标识进行比较。

优点：兼得虚拟 Cache 和物理 Cache 的好处。

局限性：Cache 容量受到限制。

【题 7.46】 答：原因：①每当进行进程切换时，由于新进程的虚拟地址(有可能与原进程的相同)所指向的物理空间与原进程的不同，故需要清空 Cache。②操作系统和用户程序对于同一个物理地址可能采用两种以上不同形式的虚拟地址来访问。它们可能会导致同一个数据在虚拟 Cache 中存在两个副本。

【题 7.47】 答：

(1) 单体多字并行存储器一次能读取 m 个指令字。如果这些指令字中有分支指令，而且分支成功，那么该分支指令之后的指令是无用的。

(2) 当前执行指令所需要的多个操作数也不一定正好都存放在同一个长存储字中。由于数据存放的随机性比程序指令存放的随机性大，所以发生这种情况的概率较大。

(3) 在这种存储器中，必须凑齐了 m 个数之后才能一起写入存储器。如果只写个别字，就必须先把相应的长存储字读出来，放到数据寄存器中，然后在地址码的控制下修改其中的一个字，最后再把长存储字写回存储器。

(4) 当要读出的数据字和要写入的数据字处于同一个长存储字内时，读和写的操作就无法在同一个存储周期内完成。

5. 应用题

【题 7.48】

解：Cache 不命中时，5×12%×2.5=1.5

(1) 考虑了 Cache 不命中时的指令平均执行时间：9.5+1.5=11(时钟周期)

(2) (11−9.5)÷9.5=15.79%

它比 Cache 命中时的指令平均执行时间延长了 15.79%。

【题 7.49】

解：(1)平均每条指令存储器停顿时钟周期数="读"的次数×读不命中率×读不命中开销+"写"的次数×写不命中率×写不命中开销=0.8×4%×25+0.5×4%×70=2.2

(2) CPU 时间=IC×[CPI+(存储停顿周期数/指令数)]×时钟周期时间

(3) 考虑 Cache 的不命中后，性能为：

CPU 时间=IC×(2.0+2.2)×时钟周期时间=IC×4.2×时钟周期时间

当考虑了 Cache 的不命中影响后，CPI 从理想计算机的 2.0 增加到 4.2，是原来的 2.1 倍。

【题7.50】

解：(1)根据题意，约75%的访存为取指令。因此，分离Cache的总体不命中率为：

$$(75\% \times 0.15\%)+(25\% \times 3.77\%)=1.055\%$$

容量为128KB的统一Cache的不命中率略低一些，只有0.95%。

(2)平均访存时间公式可以分为指令访问和数据访问两部分：

平均访存时间＝指令所占的百分比×(读命中时间＋读不命中率×不命中开销)＋数据所占的百分比×(数据命中时间＋数据不命中率×不命中开销)

所以，两种结构的平均访存时间分别为：

$$\text{分离 Cache 的平均访存时间} = 75\% \times (1+0.15\% \times 50)+25\% \times (1+3.77\% \times 50) = (75\% \times 1.075)+(25\% \times 2.885) = 1.5275$$

$$\text{统一 Cache 的平均访存时间} = 75\% \times (1+0.95\% \times 50)+25\% \times (1+1+0.95\% \times 50) = (75\% \times 1.475)+(25\% \times 2.475) = 1.725$$

因此，尽管分离Cache的实际不命中率比统一Cache的高，但其平均访存时间反而较低。分离Cache提供了两个端口，消除了结构相关。

【题7.51】

解：平均访存时间＝命中时间＋不命中率×不命中开销

① 平均访存时间＝1＋(13.34%×34)＝5.54个时钟周期

② 平均访存时间＝1＋(7.24%×34)＝3.462个时钟周期

③ 不命中开销＝30＋64÷16×2＝38

平均访存时间＝1＋(2.64%×38)＝2.003个时钟周期

④ 不命中开销＝30＋128÷16×2＝46

平均访存时间＝1＋(2.77%×46)＝2.274个时钟周期

【题7.52】

解：修改平均访存时间的公式：平均访存时间＝命中时间＋不命中率×预取命中率×1＋不命中率×(1－预取命中率)×不命中开销

则：平均访存时间＝1＋(0.64%×30%×1)＋(0.64%×(1－30%)×60)＝1.27

为了得到相同性能下的实际不命中率，我们由原始公式得：

平均访存时间＝命中时间＋不命中率×不命中开销

不命中率＝(平均访存时间－命中时间)÷不命中开销＝(1.27－1)÷60＝0.45%

【题7.53】

解：第一级Cache的不命中率(全局和局部)是110/3000，即3.67%；

第二级Cache的局部不命中率是55/110，即50%，第二级Cache的全局不命中率是55/3000，即1.83%。

【题7.54】

解：第一级Cache的全局不命中率＝它的局部不命中率＝4%

第二级Cache的全局不命中率＝第一级Cache的全局不命中率×第二级Cache局部不命中率＝4%×30%＝1.2%

第三级Cache的全局不命中率＝第一级Cache的全局不命中率×第二级Cache局部不命中率×第三级Cache局部不命中率＝4%×30%×50%＝0.6%

【题7.55】

解：平均访问时间＝命中时间＋不命中率×不命中开销

$平均访问时间_{1-路}=2.0+1.4\%\times80=3.12ns$

$平均访问时间_{2-路}=2.0\times(1+10\%)+1.0\%\times80=3.0ns$

两路组相联的平均访问时间比较小。

$$CPU_{时间}=(CPU_{执行}+存储等待周期)\times时钟周期$$

$$\begin{aligned}CPU_{时间}&=IC(CPI_{执行}+总不命中次数/指令总数\times不命中开销)\times时钟周期\\&=IC((CPI_{执行}\times时钟周期)+(每条指令的访存次数\times不命中率\times\\&\quad不命中开销\times时钟周期))\end{aligned}$$

$$CPU_{时间1-路}=IC(2.0\times2+1.2\times0.014\times80)=5.344IC$$

$$CPU_{时间2-路}=IC(2.2\times2+1.2\times0.01\times80)=5.36IC$$

$$相对性能比：\frac{CPU_{时间1-路}}{CPU_{时间2-路}}=5.36/5.344=1.003$$

直接映像Cache的访问速度比两路组相联Cache要快1.04倍，而两路组相联Cache的平均性能比直接映像Cache要高1.003倍。因此这里选择两路组相联。

【题7.56】

解：采用按写分配

(1) 写直达

Cache访问命中，有两种情况：

读命中，不访问主存；

写命中，更新Cache和主存，访问主存一次。

访问不命中，有两种情况：

读不命中，将主存中的块调入Cache中，访问主存两次；

写不命中，将要写的块调入Cache，访问主存两次，再将修改的数据写入Cache和主存，访问主存一次，共三次。

上述分析如表7.3所示。

表7.3　写直达访存情况分析

访问命中	访问类型	频　率	访存次数
Y	读	95%×75%=71.3%	0
Y	写	95%×25%=23.8%	1
N	读	5%×75%=3.8%	2
N	写	5%×25%=1.3%	3

一次访存请求最后真正的平均访存次数＝(71.3%×0)＋(23.8%×1)＋(3.8%×2)＋(1.3%×3)＝0.35

已用带宽＝$0.35\times10^9/10^9=35.0\%$

(2) 写回法

Cache访问命中，有两种情况：

读命中，不访问主存；

写命中,不访问主存。采用写回法,只有当修改的Cache块被换出时,才写入主存。

访问不命中,有一个块将被换出,也有两种情况:

如果被替换的块没有修改过,将主存中的块调入Cache块中,访问主存两次;

如果被替换的块修改过,则首先将修改的块写入主存,需要访问主存两次;然后将主存中的块调入Cache块中,需要访问主存两次,共4次访问主存,如表7.4所示。

表7.4 写回法访存情况分析

访问命中	块为脏	频 率	访存次数
Y	N	95%×70%=66.5%	0
Y	Y	95%×30%=28.5%	0
N	N	5%×70%=3.5%	2
N	Y	5%×30%=1.5%	4

所以:

一次访存请求最后真正的平均访存次数=66.5%×0+28.5%×0+3.5%×2+1.5%×4=0.13

已用带宽$=0.13\times10^9/10^9=13\%$

【题7.57】

解:(1)不管做了何种改进,不命中开销相同。不管是否交换内容,在同一"伪相联"组中的两块都是用同一个索引得到的,因此不命中率相同,即:不命中率$_{伪相联}$=不命中率$_{2路}$。

伪相联Cache的命中时间等于直接映像Cache的命中时间加上伪相联查找过程中的命中率×该命中所需的额外开销。

$$命中时间_{伪相联}=命中时间_{1路}+伪命中率_{伪相联}\times1$$

交换或不交换内容,伪相联的命中率都是由于在第一次不命中时,将地址取反,再进行第二次查找带来的。

因此,伪命中率$_{伪相联}$=命中率$_{2路}$-命中率$_{1路}$=(1-不命中率$_{2路}$)-(1-不命中率$_{1路}$)

=不命中率$_{1路}$-不命中率$_{2路}$。

交换内容需要增加伪相联的额外开销。

平均访存时间$_{伪相联}$=命中时间$_{1路}$+(不命中率$_{1路}$-不命中率$_{2路}$)×1+

不命中率$_{2路}$×不命中开销$_{1路}$

(2)将题设中的数据带入计算,得到:

$$平均访存时间_{2KB}=1+(0.098-0.076)\times1+(0.076\times50)=4.822$$

$$平均访存时间_{128KB}=1+(0.010-0.007)\times1+(0.007\times50)=1.353$$

显然128KB的伪相联Cache要快一些。

【题7.58】

解:(1)CPI=CPI$_{执行}$+存储停顿周期数/指令数

存储停顿由下列原因引起:

- 从主存中取指令
- Load和Store指令访问数据
- 由TLB引起

$$\frac{\text{存储停顿周期数}}{\text{指令数}}=\frac{\text{取指令停顿}}{\text{指令数}}+\frac{\text{数据访问停顿}+\text{TLB 停顿}}{\text{指令数}}$$

$$\frac{\text{停顿周期数}}{\text{指令数}}=\frac{\text{存储访问}}{\text{指令数}}\times\text{失效率}\times\text{失效开销}$$

$$\frac{\text{存储停顿周期数}}{\text{指令数}}=(R_{\text{指令}}P_{\text{指令}})+(f_{\text{数据}}R_{\text{数据}}P_{\text{数据}})+\frac{\text{TLB 停顿}}{\text{指令数}}$$

(2) 对于理想 TLB，TLB 不命中开销为 0。而对于统一 Cache，$R_{\text{指令}}=R_{\text{数据}}$

$P_{\text{指令}}=\text{主存延迟}+\text{传输一个块需要使用的时间}=40+32/4=48(\text{拍})$

若为读不命中，$P_{\text{数据}}=\text{主存延迟}+\text{传输一个块需要使用的时间}=40+32/4=48(\text{拍})$

若为写不命中，且块是干净的，

$P_{\text{数据}}=\text{主存延迟}+\text{传输一个块需要使用的时间}=40+32/4=48(\text{拍})$

若为写不命中，且块是脏的，

$P_{\text{数据}}=\text{主存延迟}+\text{传输两个块需要使用的时间}=40+64/4=56(\text{拍})$

$\text{CPI}=1.5+[\text{RP}+(\text{RP}\times 20\%)+0]$

指令访存全是读，而数据传输指令 Load 或 Store 指令，

$$\begin{aligned}f_{\text{数据}}\times P_{\text{数据}}&=\text{读百分比}\times(f_{\text{数据}}\times P_{\text{数据}})+\text{写百分比}\times(f_{\text{数据}}\times P_{\text{干净数据}}\times\\&\quad\text{其对应的百分比}+f_{\text{数据}}\times P_{\text{脏数据}}\times\text{其对应的百分比})\\&=20\%\times(75\%\times 48+25\%\times(50\%\times 48+50\%\times(48+16)))\\&=50(\text{拍})\end{aligned}$$

代入上述公式计算出结果如表 7.5 所示。

表 7.5　计算结果

配　　置	不命中率	CPI
16KB 直接统一映像	0.029	2.95
16KB 两路统一映像	0.022	2.6
32KB 直接统一映像	0.020	2.5

【题 7.59】

解：(1) 提高 Cache 的相联度，可以减少冲突不命中的次数，但不会影响强制性不命中和容量不命中的次数。

已知 Cache 的不命中率为 7%，程序共访问存储器 1 000 000 次，所以总的不命中次数为 70 000，其中 50% 为冲突不命中的次数。

因此，提高 Cache 的相联度能够消除的最大不命中次数是：70 000×50%=35 000。

(2) 当同时提高 Cache 的容量大小和相联度时，可以消除容量不命中和冲突不命中的次数。而这两种不命中占总不命中次数的 75%，所以，能够消除的最大不命中次数是：70 000×75%=52 500。

【题 7.60】

解：m 个分体低位交叉编址存储器的最大带宽为：

$$\text{分体数}\times\text{单体带宽}=m\times\frac{\text{存储字长}}{\text{存储周期}}=m\times\frac{4\text{B}}{2\mu\text{s}}$$

实际带宽为：

$$0.6\times 最大带宽 = 0.6\times m\times \frac{4B}{2\mu s}$$

现在要求实际带宽大于或等于4MB/s，即近似于4B/μs，故：

$$0.6\times m\times \frac{4B}{2\mu s}\geqslant 4B/\mu s$$

可得$m\geqslant 3.667$，所以主存分体数应取为4。

【题7.61】

解：因为主存容量为1M字$=2^{20}$字，所以主存地址为：20位。

Cache容量为4K字$=2^{12}$字，所以Cache地址为：12位。

因为主存由8个存储体按低位交叉编址方式组成，一个主存周期能同时对8个存储体并行访问，所以，块大小为8个字。

因此块内位移为：3位。

可得主存的块地址为20－3＝17位；Cache的块地址为12－3＝9位。

相联目录表的行数＝Cache的块数$=2^9=512$行。

相联目录表的比较位数＝主存块地址的位数＝17位。

相联目录表的宽度为：

主存块地址位数＋Cache块地址位数＋有效位位数＝17＋9＋1＝27位。

相联目录表总位数为：

行数×宽度＝512×27＝13 824位。

【题7.62】

解：每个存储周期能访问到的平均字数为：

$$B=\frac{1-(1-\lambda)^m}{\lambda}$$

已知$\lambda=25\%$，$m=32$，可得：

$$B=\frac{1-(1-\lambda)^m}{\lambda}=\frac{1-0.75^{32}}{0.25}\approx 4$$

即每个存储周期能访问到4个字。

如果$\lambda=25\%$，$m=16$，可得：

$$B=\frac{1-(1-\lambda)^m}{\lambda}=\frac{1-0.75^{16}}{0.25}\approx 3.96$$

即每个存储周期能访问到3.96个字。

由此可以看出，当转移概率为25%，比较大时，采用模32和模16的每个存储周期能访问到的平均字数非常相近。就是说，此时靠提高模数m来提高主存实际带宽的效果不大。

第8章　输入输出系统

8.1　基本要求与难点

8.1.1　基本要求

(1) 掌握有关输入输出系统的基本概念。

(2) 理解I/O系统的性能参数，理解存储外设的可靠性参数：可靠性、可用性和可信性。

(3) 理解廉价磁盘冗余阵列RAID的相关概念，掌握各级RAID的结构、工作原理和特点。

(4) 理解总线设计所需考虑的问题，了解若干总线标准和实例，了解总线与CPU的连接。

(5) 理解通道的作用和功能，熟练掌握通道的工作过程。

(6) 掌握通道的分类以及各类通道的工作原理和特点。

(7) 能熟练地进行通道的流量分析。

(8) 了解DMA对虚拟存储器的影响，了解I/O对Cache数据一致性的影响。

8.1.2　难点

(1) 各级RAID的结构、工作原理和特点。

(2) 各类通道的工作原理和特点。

(3) 通道的流量分析。

8.2　知识要点

8.2.1　I/O系统的性能

I/O系统的性能对CPU的性能有很大的影响，若两者的性能不匹配，I/O系统就有可能成为整个系统的瓶颈。系统的响应时间是衡量系统性能的一个很好的指标。它是指从用户输入命令开始，到得到结果所花的时间。这个时间由两部分构成：I/O系统的响应时间以及CPU的处理时间。如果I/O系统的响应时间很长，CPU再快也没用。

评价I/O系统性能的参数主要有：**连接特性**，**I/O系统的容量**，**响应时间**和**吞吐率**等。**连接特性**是指哪些I/O设备可以和计算机系统相连接，**I/O系统的容量**是指I/O系统可以容纳的I/O设备数。

8.2.2　I/O系统的可靠性、可用性和可信性

系统的可靠性是指系统从某个初始参考点开始一直连续提供服务的能力，它通常用**平均无故障时间**(Mean Time To Failure，MTTF)来衡量。MTTF的倒数就是系统的失效率。如果系统中每个模块的生存期服从指数分布，则系统整体的失效率是各部件的失效率之和。系统中断服务的时间用**平均修复时间**(Mean Time To Repair，MTTR)来衡量。

系统的可用性是指系统正常工作的时间在连续两次正常服务间隔时间中所占的比率。

$$可用性 = \frac{MTTF}{MTTF + MTTR} \tag{8.1}$$

式(8.1)中的MTTF+MTTR通常可以用**平均失效间隔时间**(Mean Time Between Failures，MTBF)来代替。

系统的**可信性**是指服务的质量，即在多大程度上可以合理地认为服务是可靠的。可信性与可靠性和可用性不同，它是不可以度量的。

提高系统组成部件的可靠性的方法包括有效构建方法和纠错方法。**有效构建**是指在构建系统的过程中消除故障隐患，这样建立起来的系统就不会出现故障。**纠错方法**是指在系统构建中采用容错的方法。这样即使出现故障，也可以通过容错信息保证系统正常工作。

8.2.3　廉价磁盘冗余阵列 RAID

磁盘阵列(Disk Array，DA)是使用多个磁盘(包括驱动器)的组合来代替一个大容量的磁盘。这不仅能比较容易地构建大容量的磁盘存储器系统，而且可以提高系统的性能。磁盘阵列一般是以条带为单位把数据均匀地分布到多个磁盘上(交叉存放)。条带存放使得磁盘存储器系统可以并行地处理多个数据读/写请求，从而提高总的I/O性能。

我们可以通过在磁盘阵列中设置冗余信息盘来解决可靠性下降的问题。当单个磁盘失效时，丢失的信息可以通过冗余盘中的信息重新构建。由于磁盘的平均无故障时间MTTF为几十年，而平均修复时间MTTR只有几个小时，所以容错技术使得磁盘阵列的可靠性比单个磁盘高很多。这种磁盘阵列被称为**RAID**，即**廉价磁盘冗余阵列**(Redundant Array of Inexpensive Disks)，有些书里也将其称为**独立磁盘冗余阵列**(Redundant Array of Independent Disks)。

大多数磁盘阵列的组成可以用以下两个特征来区分。

(1) 数据交叉存放的粒度；

(2) 冗余数据的计算方法以及在磁盘阵列中的存放方式。

数据交叉存放的粒度有细粒度和粗粒度之分。**细粒度磁盘阵列**是在概念上把数据分割成相对较小的单位交叉存放。这样几乎所有的I/O请求，不管大小，都会访问磁盘阵列中的所有磁盘。其结果是所有I/O请求都能够获得很高的数据传输率。其缺点是在任何时

间，都只有一个逻辑上的 I/O 在处理当中，而且所有的磁盘都会因为为每个请求进行定位而浪费时间。

粗粒度磁盘阵列是把数据以相对较大的单位交叉存放。这样规模较小的 I/O 请求只需要访问数量较少的几个磁盘，只有较大规模的请求才会访问到所有的磁盘。多个较小规模的请求可以同时得到处理，而对于较大规模的请求来说又能获得较高的传输率。

在冗余信息的计算方面，当今的磁盘阵列大多都是采用奇偶校验码，但也有采用汉明码或 Reed-Solomon 码的。在冗余信息的存放方式方面，有两种方法：①把冗余信息集中存放在少数的几个盘中；②把冗余信息均匀地存放到所有的盘中。

在 RAID 中增加冗余信息盘有几种不同的方法，它们构成了不同的 RAID 级别，如表 8.1 所示。各级 RAID 的代价和性能各不相同。

表 8.1　RAID 的分级及其特性

RAID 级别	可以容忍的故障个数以及当数据盘为 8 个时，所需要的检测盘的个数	优点	缺点	公司产品
0 非冗余，条带存放	0 个故障； 0 个检测盘	没有空间开销	没有纠错能力	广泛应用
1 镜像	1 个故障； 8 个检测盘	不需要计算奇偶校验，数据恢复快，读数据快。而且其小规模写操作比更高级别的 RAID 快	检测空间开销最大（即需要的检测盘最多）	EMC，HP (Tandem)，IBM
2 存储器式 ECC	1 个故障； 4 个检测盘	不依靠故障盘进行自诊断	检测空间开销的级别是 $\log_2 m$ 级（m 为数据盘的个数	没有
3 位交叉奇偶校验	1 个故障； 1 个检测盘	检测空间开销小（即需要的检测盘少），大规模读写操作的带宽高	对小规模、随机的读写操作没有提供特别的支持	外存概念
4 块交叉奇偶校验	1 个故障； 1 个检测盘	检测空间开销小，小规模的读操作带宽更高	校验盘是小规模写的瓶颈	网络设备
5 块交叉分布奇偶校验	1 个故障； 1 个检测盘	检测空间开销小，小规模的读写操作带宽更高	小规模写操作需要访问磁盘 4 次	广泛应用
6P+Q 双奇偶校验	2 个故障； 2 个检测盘	具有容忍两个故障的能力	小规模写操作需要访问磁盘 6 次，检测空间开销加倍（与 RAID3、4、5 比较）	网络设备

1. RAID0

RAID0 是非冗余磁盘阵列。它是 RAID 中最简单的一种，实现成本也最低。它把数据

切分成**条带**(strip),以条带为单位交叉地分布存放到多个磁盘中。当从磁盘阵列中按顺序读取这些数据时,所有的磁盘都可以并行工作,各自读出相应的部分。因此其性能很高。但由于RAID0不提供数据冗余,因此一旦数据被损坏,将无法得到恢复。

2. RAID1

RAID1是最基本的一种冗余磁盘阵列,称为**镜像磁盘**。其核心思想是对所有的磁盘数据提供一份冗余的备份。每当把数据写入磁盘时,都要将该数据也写入其镜像盘。因而在系统中所有的数据都有两份。

优点:简单,且能实现快速的读取操作。其速度比级别更高的RAID都快。数据的恢复也很简单,不需要进行数据重建计算。

缺点:所需要的磁盘的总数是采用镜像前磁盘个数的两倍。实现成本最高。

3. RAID2

RAID2是**存储器式的磁盘阵列**。之所以有这样的名称,是因为它是按Hamming纠错码的思路来构建的。每个数据盘存放所有数据字的一位,按位交叉存放。而每当从数据盘读出数据时,把其Hamming码也读出来,用于判断数据是否有错。

虽然在RAID的分级上有这一级,但实际上并没有商业化的产品。

4. RAID3

RAID3即**位交叉奇偶校验磁盘阵列**。

因为当某个磁盘出故障时,磁盘控制器本身能够很容易地发现是哪个磁盘出错,所以采用奇偶校验就够了。故设置一个校验盘,专门用于存放数据盘中相应数据的奇偶校验。在数据写入磁盘时,为每行数据形成奇偶校验位并写入校验盘。在读出数据时,如果控制器发现某个磁盘出故障,就可以根据故障盘以外的所有其他盘中的正确信息恢复故障盘中的数据,这是通过异或运算来实现的。即使故障盘为校验盘,也可以照此处理。

这是一种细粒度的磁盘阵列,即采用的条带宽度较小,甚至可以是1个字节或1位。

优点:由于是细粒度的,所以对于绝大多数的I/O请求,都需要磁盘阵列中的所有磁盘为之服务,因而能够获得很高的数据传输率。这种磁盘阵列对大数据量的读写具有很大的优越性。另外,不管数据盘有多少个,RAID3只需要一个校验盘,校验空间开销比较小。

缺点:不能同时进行多个I/O请求的处理,对多个小规模的I/O请求来说表现较差。

5. RAID4

RAID4是**块交叉奇偶校验磁盘阵列**。

许多应用程序中磁盘读写都是小规模的访问。对于这些应用来说,磁盘阵列最好能同时处理多个小规模访问请求。采用粗粒度的磁盘阵列就能实现这一目标,即采用比较大的条带,以块为单位进行交叉存放和计算奇偶校验,这就是RAID4。

优点:每个磁盘能够独立地进行读操作,从而提高单位时间完成的读操作的数量。对于读取操作,每次只需访问数据所在的磁盘。仅在磁盘出现故障时,才会去读校验盘,并进

行数据的重建。RAID4 除了能有效地处理小规模访问，而且能跟 RAID3 一样快速地处理大规模访问。此外，RAID4 的校验空间开销也比较小，只需要一个校验盘。

缺点：对于写入操作，由于要重新计算校验码，所以要对多个磁盘进行读和写的操作。这对磁盘访问的速度有较大的影响。而且系统中只有一个校验盘，它很容易成为瓶颈。

6. RAID5

RAID5 是**块交叉分布奇偶校验磁盘阵列**。

在 RAID4 中，所有的写入操作都必须读和写校验盘。而系统中校验盘只有一个，很容易成为瓶颈。RAID5 通过把校验信息分布到磁盘阵列中的各个磁盘来解决这个问题。这里每一行数据块的校验块被依次错开、循环地存放到不同的盘中，以达到均匀分布的目的。

优点：除了能跟 RAID3 一样快地处理大规模访问、能跟 RAID4 一样快地处理小规模读操作以外，还能比它们都更快地处理小规模写操作。而且校验空间开销比较小，也是只需要一个校验盘。

缺点：其控制器是经典 RAID(RAID1～RAID5)中最复杂的了。

7. RAID6

RAID6 是 **P+Q 双校验磁盘阵列**。它是在 RAID5 的基础上增加了一个独立的校验信息，放在另一个校验盘中。

优点：能够容忍两个磁盘出错，很适合于重要数据的保存。

缺点：所需的校验空间开销是 RAID5 的两倍。写操作要访问磁盘的次数比较多。

8. RAID10 与 RAID01

它们都是 RAID0 与 RAID1 相结合的结果，其区别在于先做什么。假设共有 8 个磁盘，那么由于需要镜像，就相当于只有 4 个盘可用来存放数据，另外 4 个盘是作为镜像盘。RAID1+0 是先进行镜像(RAID1)，然后再进行条带存放(RAID0)，即把盘分为 4 组，组内做镜像，组之间按条带存放。RAID0+1 是先进行条带存放(RAID0)，然后再进行镜像(RAID1)，即组内按条带存放，组之间做镜像。

8.2.4　总线

总线的优点是简单、成本低。其主要缺点在于它是由不同的外设分时共享的，形成了信息交换的瓶颈，从而限制了系统中总的 I/O 吞吐量。

1. 总线的设计

与计算机中其他子系统的设计一样，总线的设计取决于需要达到的性能和实现成本。表 8.2 给出了设计总线时需要考虑的一些问题。

表 8.2　总线的主要特性

特　　性	高　性　能	低　价　格
总线宽度	独立的地址和数据总线	数据和地址分时共用同一套总线
数据总线宽度	越宽越快(例如：64位)	越窄越便宜(例如：8位)
传输块大小	块越大总线开销越小	单字传送更简单
总线主设备	多个(需要仲裁)	单个(无须仲裁)
分离事务	采用,因为分离的请求包和回答包能提高总线带宽	不采用,因为持续连接成本更低,而且延迟更小
定时方式	同步	异步

显然,表中的前三点的观点是非常明确的。采用独立的地址和数据线、更宽的数据总线以及成块的数据传输都将提高总线的性能。当然,这同时也带来了成本的提高。

在有多个主设备的情况下,如果不采用传统的持续占用总线的方法,而改用包交换,就能提高总线带宽。其基本思想是将总线事务分成请求和应答两部分。在请求和应答之间的空闲时间内,总线可以供给其他的I/O使用。当发出读请求、存储器根据地址去读数据字时,分离事务总线被释放,允许其他主设备使用。采用这种技术的总线称为分离事务总线,也称为包交换总线等。分离事务总线有较高的带宽,但是它的数据传送延迟通常比独占总线方法的大。

2. 总线标准和实例

I/O总线是计算机系统中连接设备的接口。制定总线标准是非常重要的,因为只要计算机和I/O设备的设计都满足相同的标准,那么任意一台I/O设备就可以与任意一台计算机相连接。I/O总线标准就是定义如何将设备与计算机进行连接的文档。

常见的并行I/O总线有：IDE/Ultra ATA、SCSI、PCI、PCI X等；常见的串行I/O总线有：I^2C、1-wire、RS-232、SPI等。

3. 与CPU的连接

I/O总线的物理连接方式有两种选择,一种是连接到存储器上,另一种是连接到Cache上。一般来说,前者更常见一些。在一些低成本的系统中,I/O总线往往就是CPU主存总线,此时总线上的I/O命令将影响CPU的访存,例如取指令。

CPU对I/O设备的编址有两种方式。最常用的方式是"**存储器映射I/O**",也称为**I/O设备统一编址方式**。在这种方法中,将一部分存储器地址空间分配给I/O设备,用load指令和store指令对这些地址进行读写将引起I/O设备的输入和输出操作。

另一种I/O设备的编址方式是**I/O设备独立编址**,并在CPU中设置专用的I/O指令来访问它们。

8.2.5　通道处理机

1. 通道的作用和功能

为了把对外设的管理工作从CPU中分离出来,使CPU摆脱繁重的输入/输出负担,也

为了使设备能共享输入/输出接口，从 IBM360 系列机开始，普遍采用了通道技术。即由一种称为通道的专用处理机来专门负责整个计算机系统的输入/输出工作。通道处理机只能执行有限的一组输入输出指令。

大型计算机系统可以有多个通道，每个通道可以连接多个设备控制器，而每个控制器又可以管理一台或多台外设。这样就形成了一个典型的由 CPU、通道、设备控制器、外设构成的 4 级层次结构的输入/输出系统。

一般来说，通道的功能包括以下几个方面。

(1) 接收 CPU 发来的 I/O 指令，并根据指令要求选择指定的外设与通道相连接。

(2) 执行通道程序。并根据需要向被选中的设备控制器发出各种操作命令。

(3) 给出外设中相关数据所在的地址。如磁盘存储器的柱面号、磁头号、扇区号等。

(4) 给出主存缓冲区的首地址。

(5) 控制外设与主存缓冲区之间的数据传送的长度。即对传送的数据个数进行计数，并判断数据传送是否结束。

(6) 指定传送工作结束时要进行的操作。例如，将外设的中断请求及通道的中断请求送往 CPU 等。

(7) 检查外设的工作状态是否正常，并将该状态信息送往主存指定单元保存。

(8) 在数据传输过程中完成必要的格式变换，例如，把字拆分为字节。

通道的主要硬件如下。

(1) 寄存器：数据缓冲寄存器，主存地址计数器，传输字节数计数器，通道命令字寄存器，通道状态字寄存器等。

(2) 控制逻辑：分时控制，地址分配，数据传送，数据装配和拆分等。

通道对外设的控制通过输入/输出接口和设备控制器进行。通道与设备控制器之间一般采用标准的输入/输出接口来连接。通道通过标准接口把操作命令送到设备控制器，设备控制器解释并执行这些通道命令，完成命令指定的操作。设备控制器能够记录外设的状态，并把状态信息送往通道和 CPU。

2. 通道的工作过程

用户通过调用通道来完成一次数据输入输出的过程如图 8.1 所示，CPU 执行程序和通道执行通道程序的时间关系如图 8.2 所示。

利用通道完成一次数据传输的主要过程分为以下三步。

(1) 在用户程序中使用访管指令进入管理程序，由管理程序来编制一个通道程序，并启动通道。

(2) 通道处理机执行通道程序，完成指定的数据输入/输出工作。

(3) 通道程序结束后向 CPU 发中断请求。

数据传送完成后，通道向 CPU 发 I/O 中断请求。CPU 响应该中断请求，再次进入管态，调用相应的管理程序对该中断请求进行处理。然后，CPU 返回到目态，继续进行目态程序的执行。

这样，在每一次输入/输出中，CPU 只需要两次调用管理程序，大大减少了对用户程序的打扰。当系统中有多个通道同时工作时，CPU 与多种不同类型、不同速度的外设可以充

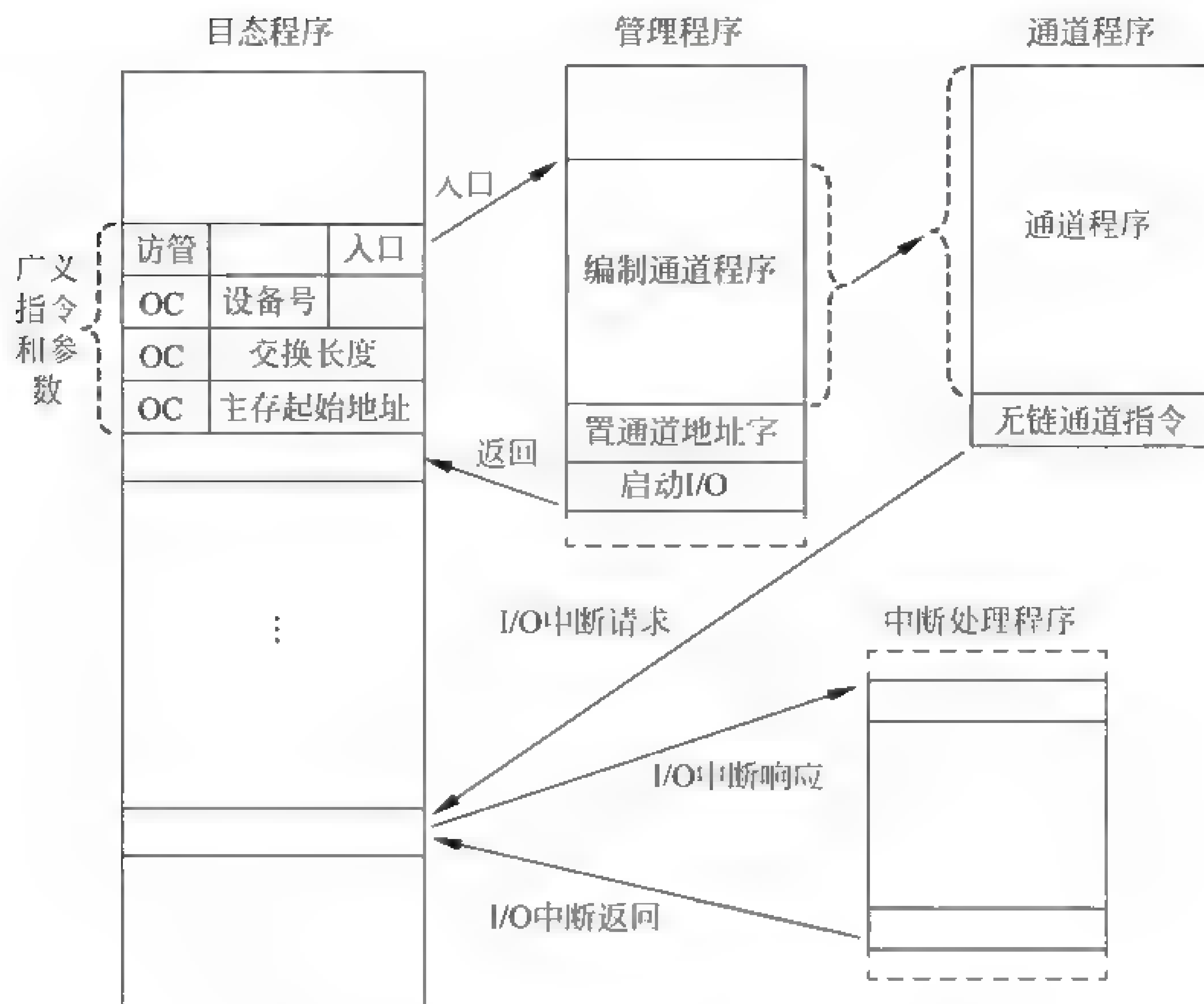


图 8.1　通道来完成一次数据输入输出的过程

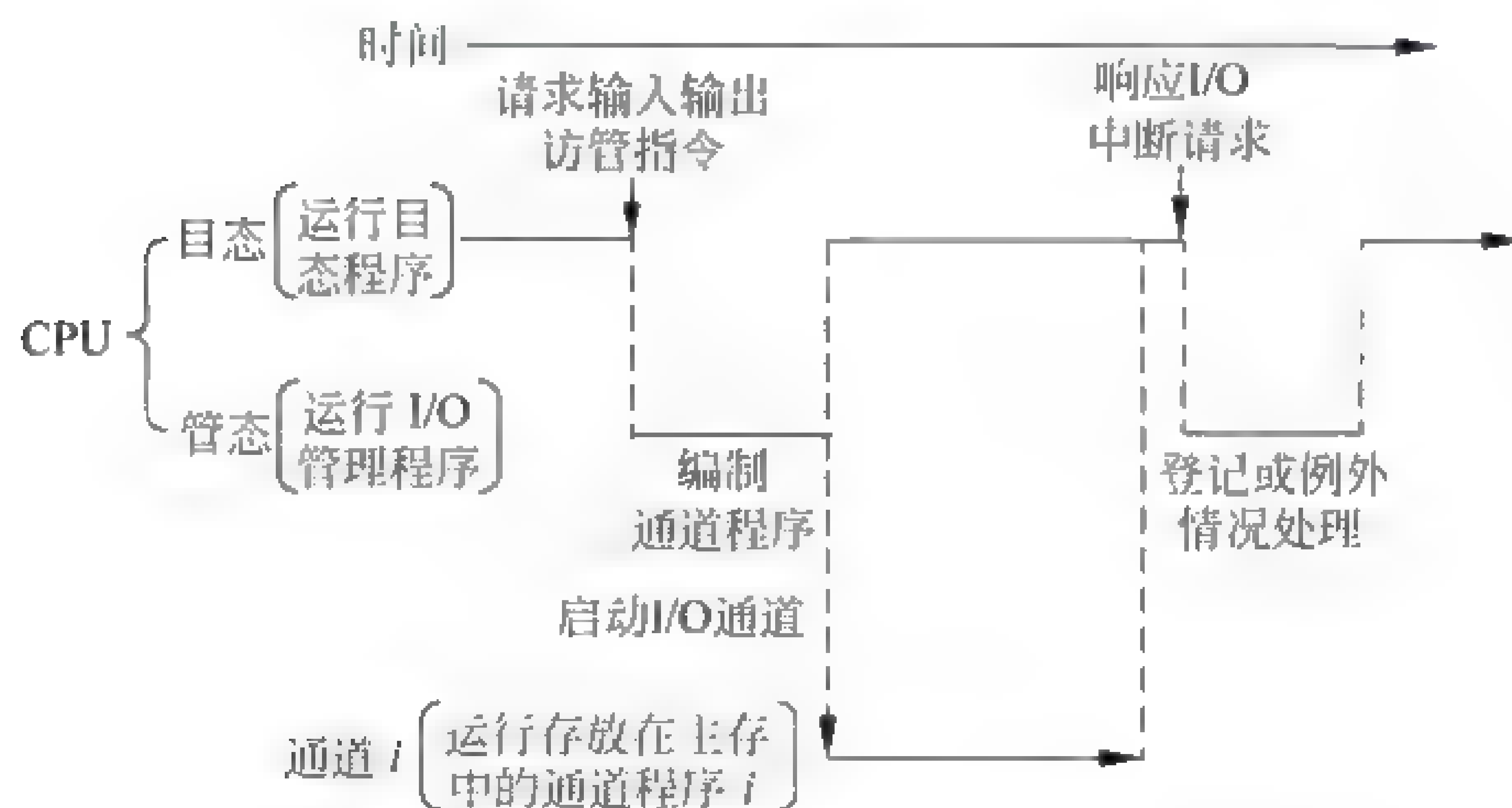


图 8.2　CPU 执行程序和通道执行通道程序的时间关系

分并行地工作。

3. 通道种类

1) 字节多路通道

字节多路通道是一种简单的共享通道,用于连接多台低速或中速的设备。这些设备一般是以字节为宽度进行输入/输出,而且相邻的两次传送之间有较长时间的等待。字节多路通道以字节交叉的方式分时轮流地为它们服务。

2) 选择通道

像磁盘存储器这样的高速外设需要很高的数据传输率,因此应该设置专门的通道,在一段时间内只为一台高速外设独占使用。这就是选择通道。当有多台高速设备请求传送数据

时，选择通道是按照一定的规则对要服务的设备进行选择。一旦选中某一设备，通道就进入“忙”状态，直到该设备的数据传输工作全部完成为止。处理完后，再重新选择。

3）数组多路通道

数组多路通道可以看成是字节多路通道与选择通道相结合的产物。与选择通道一样，它也是适用于高速设备，但它不像选择通道那样一次就把所选设备的数据全部传送完，而是在传送完固定长度的一个数据块（例如 512 字节）之后，就重新选择别的设备。因此它是以数据块为单位、分时轮流地为多台高速设备提供服务。

数组多路通道之所以能够并行地为多台高速设备服务，是因为虽然其所连设备的传输速率很高，但其寻址等辅助操作时间很长。例如，磁盘存储器的寻址时间要比数据传输时间长两个数量级以上。数组多路通道在向一台高速设备发出定位命令后，就立即从逻辑上与该设备断开，直到定位完成时再进行连接，发出找扇区命令后再一次断开，直到开始数据传送。因此，数组多路通道的实际工作方式是：通道在为一台高速设备传送数据时，有多台高速设备可以在定位或者在找扇区。

4. 通道流量分析

通道流量是指一个通道在数据传送期间，单位时间内能够传送的数据量，所用单位一般为字节/秒。通道流量也称为通道吞吐率，通道数据传输率等。一个通道在满负荷工作状态下的流量称为通道最大流量。通道最大流量主要与通道的种类、通道选择设备一次所用的时间以及传送一个字节所用的时间等因素有关。

为便于讨论，下面给出一些后面要用到的参数的定义。

（1）T_S：设备选择时间。从通道响应设备发出的数据传送请求开始，到通道实际为这台设备传送数据所需要的时间。

（2）T_D：传送一个字节所用的时间。

（3）p：在一个通道上连接的设备台数，且这些设备同时都在工作。

（4）n：每台设备传送的字节数，这里假设每台设备传送的字节数都相同。

（5）k：数组多路通道传输的一个数据块中包含的字节数。在一般情况下，$k<n$。对于磁盘、磁带等磁表面存储器，通常 $k=512$。

（6）T：通道完成全部数据传送工作所需要的时间。

1）字节多路通道

字节多路通道是分时为多台低速和中速外设服务的。通道每连接一台外设，只传送一个字节，然后又选择与另一台设备连接，并传送一个字节，如此反复。p 台设备每台传送 n 个数据总共所需的时间为：

$$T_{BYTE} = (T_S + T_D) \times p \times n \tag{8.2}$$

其最大流量为

$$f_{MAX\text{-}BYTE} = \frac{pn}{(T_S + T_D)pn} = \frac{1}{T_S + T_D} \tag{8.3}$$

根据字节多路通道的工作原理可知，它的实际流量是连接在这个通道上的所有设备的数据传输率之和，即

$$f_{BYTE} = \sum_{i=1}^{p} f_i \tag{8.4}$$

其中，f_i为第 i 台设备的实际数据传输率。下同。

2）选择通道

选择通道在一段时间内只能单独为一台高速外设服务，当这台设备的数据传送工作全部完成后，通道才能为另一台设备服务。选择通道实际上是逐个为物理上连接的几台高速外设服务的。

p 台设备每台传送 n 个数据总共所需的时间为：

$$T_{\text{SELECT}} = pT_S + pnT_D \tag{8.5}$$

其最大流量为

$$T_{\text{MAX-SELECT}} = \frac{pn}{pT_S + pnT_D} = \frac{1}{\frac{T_S}{n} + T_D} \tag{8.6}$$

3）数组多路通道

通道每连接一台高速设备，传送一个数据块(设为 k 个字节)，传送完成后，又与另一台高速设备连接，再传送一个数据块……p 台设备每台传送 n 个数据总共所需的时间为：

$$T_{\text{BLOCK}} = \frac{pnT_S}{k} + pnT_D \tag{8.7}$$

其最大流量为

$$T_{\text{MAX-BLOCK}} = \frac{pn}{\frac{pnT_S}{k} + pnT_D} = \frac{1}{\frac{T_S}{k} + T_D} \tag{8.8}$$

对于选择通道和数组多路通道，在一段时间内，一个通道只能为一台设备传送数据。此时通道的实际流量就等于该设备的数据传输率。因此，这两种通道的实际流量就是连接在这个通道上的所有设备中数据流量最大的那一个。

$$\begin{aligned} f_{\text{BLOCK}} &\leqslant \max_{i=1}^{p} f_i \\ f_{\text{SELECT}} &\leqslant \max_{i=1}^{p} f_i \end{aligned} \tag{8.9}$$

为了保证通道能够正常工作而不丢失数据，各种通道的实际流量应该不大于通道的最大流量，即应该满足下列不等式：

$$\begin{aligned} f_{\text{BYTE}} &\leqslant f_{\text{MAX-BYTE}} \\ f_{\text{BLOCK}} &\leqslant f_{\text{MAX-BLOCK}} \\ f_{\text{SELECT}} &\leqslant f_{\text{MAX-SELECT}} \end{aligned} \tag{8.10}$$

两边的差值越小，通道的利用率就越高。当两边相等时，通道处于满负荷工作状态。为防止数据丢失，在实际设计通道的最大流量时，还应留有一定的余量。

8.2.6 I/O与操作系统

1. DMA和虚拟存储器

在具有虚拟存储器的计算机中，存在DMA是使用虚拟地址还是物理地址来传输数据的问题。如果使用物理地址进行DMA传输，存在以下两个问题。

(1) 当数据缓冲区大小超过一页时，由于缓冲区所使用的多个页面在物理存储器中不一定是连续的，所以传输可能会出现问题。

(2) 如果 DMA 正在存储器和缓冲区之间传输数据时，操作系统从存储器中移出（或重定位）一些页面，那么，DMA 将会在存储器中错误的物理页面上进行数据传输。

解决这些问题的一种方法是使操作系统在 I/O 的传输过程中确保 DMA 设备所访问的页面都位于物理存储器中，这些页面被称为是钉(pinned)在了主存中。

另一种方法是采用“**虚拟 DMA**”技术，它允许 DMA 设备直接使用虚拟地址，并在 DMA 传送的过程中由硬件将虚拟地址转换为物理地址。这样，I/O 使用的缓冲区在虚拟地址空间中是连续的，但在物理存储器中是可以分散存放的。

在采用虚拟 DMA 的情况下，如果进程在内存中被移动，操作系统应该能够及时地修改相应的 DMA 地址表。

2. I/O 和 Cache 数据一致性

对主存的直接 I/O 操作可能会导致 Cache 内容与主存内容不一致的问题。如果把 I/O 直接连接到 Cache 上，则不会产生由 I/O 导致的数据不一致的问题。在这种情况下，所有 I/O 设备和 CPU 都能在 Cache 中看到最新的数据。但这种方法会造成对 CPU 的较多的打扰。因此，许多计算机还是选择把 I/O 直接连接到存储器上，把存储器的一片区域作为 I/O 缓冲器，并设法解决数据不一致的问题。

可以通过以下方法来解决内容的一致性问题。

1) 软件的方法

设法保证 I/O 缓冲器中的所有各块都不在 Cache 中。具体做法有两种，一种是把 I/O 缓冲器的页面设置为不可进入 Cache 的，在进行输入操作时，操作系统总是把输入的数据放到该页面。另一种方法是在进行输入操作之前，操作系统先把 Cache 中与 I/O 缓冲器相关的数据“赶出”Cache，即把相应的数据块设置为“无效”状态。当然，如果是采用写回法的 Cache，而且该数据块被修改过，就还需要把整块的内容写回存储器。

2) 硬件的方法

硬件的方法是在进行输入操作时，检查相应的 I/O 地址（I/O 缓冲器中的单元）是否在 Cache 中。这是通过把这些 I/O 地址与 Cache 中的标识进行比较来判断的。为了使这些比较能与 CPU 对 Cache 的访问并行进行，可以多设置一套完全相同的标识存储器。如果发现 I/O 地址在 Cache 中有匹配的，就把相应的 Cache 块设置为“无效”。

习　　题

1. 概念题

【题 8.1】 解释以下名词

响应时间	可靠性	可用性	可信性
RAID	分离事务总线	通道	通道流量
虚拟 DMA			

2. 选择题

【题 8.2】 输入输出数据不经过 CPU 内部寄存器的输入输出方式是(　　)。

A. 程序控制输入输出方式　　B. 中断输入输出方式

C. 直接存储器访问方式　　D. 上述 3 种方式

【题 8.3】 在配置有通道的计算机系统中,用户程序需要输入输出时,引起的中断是(　　)。

A. 访管中断　　B. I/O 中断　　C. 程序性中断　　D. 外部中断

【题 8.4】 当计算机系统通过执行通道程序完成输入输出工作时,执行通道程序的是(　　)。

A. CPU　　B. 通道　　C. CPU 和通道　　D. 指定的外设

【题 8.5】 磁盘存储器适合于连接到(　　)。

A. 字节多路通道或选择通道　　B. 字节多路通道或数组多路通道

C. 选择通道或数组多路通道　　D. 任意一种通道

【题 8.6】 在由多个通道组成的 I/O 系统中,I/O 系统的最大流量是(　　)。

A. 各通道最大流量的最大值　　B. 各通道最大流量之和

C. 各通道实际流量的最大值　　D. 各通道实际流量之和

3. 填空题

【题 8.7】 输入/输出系统包括________和________。

【题 8.8】 评价 I/O 系统性能的参主要有________、________、________和________。

【题 8.9】 MTTR 是指________,MTTF 是指________。

【题 8.10】 磁盘阵列中,数据交叉存放的粒度有________和________之分。前者是在概念上把数据分割成相对________的单位交叉存放,而后者是把数据以相对________的单位交叉存放。

【题 8.11】 实现盘阵列的方式主要有________、________和________三种。

【题 8.12】 按用途分类,总线可分为________总线和________总线。

【题 8.13】 按设备定时方式分类,总线可分为________和________。

【题 8.14】 I/O 总线的物理连接方式有两种选择,一种是连接到________上,另一种是连接到________上。

【题 8.15】 通过通道完成一次输入输出工作,CPU 需要两次调用操作系统的管理程序,第一次调用是为了________,第二次调用是为了________。

【题 8.16】 通道的主要硬件包括________和________两部分。

【题 8.17】 通道分为________、________和________3 种类型。

【题 8.18】 字节多路通道实际流量是________;数组多路通道的实际流量是________。

【题 8.19】 在通道方式 I/O 传输过程中,用户经由________指令来使用外设。进管后________要编制________。CPU 在执行完________指令后,通道就可以与其并行工作。

【题 8.20】 若 T_S 是通道的设备选择时间,T_D 是通道传送一个字节数据所需的时间,p

为通道连接的设备台数，且这些设备同时都在工作。n 为每台外设需要传送的字节数。K 为数组多路通道传输的一个数据块中包含的字节数。那么，字节多路通道完成数据传送的时间 $T_{byte}=$________，选择通道完成数据传送的时间 $T_{select}=$________，数组多路通道完成数据传送的时间 $T_{block}=$________。

【题 8.21】 为保证通道能够正常工作，通道的________流量应该大于或等于通道的________流量。

4. 问答题

【题 8.22】 简述提高系统组成部件的可靠性的两种方法。

【题 8.23】 RAID 有哪些分级？各有何特点？

【题 8.24】 大多数磁盘阵列的组成可以用哪两个特征来区分？

【题 8.25】 同步总线和异步总线各有什么优缺点？

【题 8.26】 简述通道的主要功能。

【题 8.27】 简述通道完成一次数据传输的主要过程。

【题 8.28】 通道分为哪 3 种类型？它们分别为哪种外围设备服务？

【题 8.29】 简述字节多路通道、数组多路通道和选择通道的数据传送方式。

【题 8.30】 在有 Cache 的计算机系统中，进行 I/O 操作时，会产生哪些数据不一致问题？如何克服？

5. 应用题

【题 8.31】 计算机系统字长 32 位，包含两个选择通道和一个多路通道，每个选择通道上连接了两台磁盘机和两台磁带机，多路通道上连接了两台行式打印机，两台读卡机，10 台终端，假定各设备的传输率如下。

磁盘机：800KB/s

磁带机：200KB/s

行打机：6.6KB/s

读卡机：1.2KB/s

终端：1KB/s

计算该计算机系统的最大 I/O 数据传输率。

【题 8.32】 设某个数组多路通道设备选择时间 T_S 为 1μs，传送一个字节数据所需的时间 T_D 为 1μs，一次传送定长数据块的大小 k 为 512B。现有 8 台外设的数据传输速率分别如表 8.3 所示，问哪些外设可连接到该通道上正常工作？

表 8.3　8 台外设的数据传输速率

设备名称	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8
数据传输速率	1000	480	480	800	512	512	1024	1024

【题 8.33】 有 8 台外设的数据传输速率分别如表 8.4 所示，现设计一种通道，通道可实现设备选择时间 $T_S=2\mu s$，数据传送时间 $T_D=2\mu s$。

(1) 如果按字节多路通道设计,该通道的最大流量是多少?若希望从8台外设中至少选择4台外设同时连接到该通道上,而且尽量多连接传输速率高的设备,那么,应选择哪些外设连接到该通道上?

(2) 如果按数组多路通道设计,且一次传送定长数据块的大小 $k=512$B,该通道的最大流量是多少?从8台外设中可选择哪些外设连接到该通道上?

表 8.4 外设的数据传输速率

设 备 号	1	2	3	4	5	6	7	8
数据传输速率	500	240	100	75	50	40	14	10

【题 8.34】 设某个字节多路通道的设备选择时间 T_S 为 9.8μs,传送一个字节的数据所需的时间 T_D 为 0.2μs。若某种低速外设每隔 500μs 发出一次传送请求,那么,该通道最多可连接多少台这种外设?

【题 8.35】 一个字节多路通道连接有6台设备,它们的数据传输率如表8.5所示。

表 8.5 数据传输率

设备名称	D_1	D_2	D_3	D_4	D_5	D_6
数据传输速率/(B/ms)	50	50	40	25	25	10

(1) 计算该通道的实际工作流量。

(2) 若通道的最大流量等于实际工作流量,求通道的工作周期 T_S+T_D。

【题 8.36】 通道型I/O系统由一个字节多路通道A、两个数组多路通道B1和B2以及一个选择通道C组成。其中,通道A连接两个子通道A1和A2,A1和A2分别连接8台低速外设,B1、B2和C分别连接5台高速外设。各外设的数据传输速率如表8.6所示。

(1) 分别计算通道A、B1、B2和C的最大流量至少为多大才不会丢失传送的数据?

(2) 若I/O系统的流量为主存带宽的1/2,则主存带宽应达到多大?

表 8.6 各外设的数据传输速率

通 道		通道连接的各外设的数据传输速率							
字节多路通道 A	子通道 A1	64	56	56	20	20	20	10	10
	子通道 A2	56	30	30	30	30	30	30	20
数组多路通道 B1		512	512	512	256	256			
数组多路通道 B2		256	256	512	256	512			
选择通道 C		1024	512	512	512	512			

【题 8.37】 假设在一个计算机系统中:

(1) 每页为32KB,Cache块大小为128B。

(2) 对应新页的地址不在Cache中,CPU不访问新页中的任何数据。

(3) Cache中95%的被替换块将再次被读取,并引起一次不命中。

(4) Cache使用写回方法,平均60%的块被修改过。

(5) I/O系统缓冲能够存储一个完整的Cache块。

（6）访问或不命中在所有 Cache 块中均匀分布。

（7）在 CPU 和 I/O 之间，没有其他访问 Cache 的干扰。

（8）无 I/O 时，每 100 万个时钟周期内有 18 000 次不命中。

（9）不命中开销是 40 个时钟周期。如果被替换的块被修改过，则再加上 30 个周期用于写回主存。

（10）假设计算机平均每 200 万个周期处理一页。

试分析 I/O 对于性能的影响有多大。

题　　解

1. 概念题

【题 8.1】　解释以下名词

响应时间——从用户输入命令开始，到得到结果所花的时间。

可靠性——指系统从某个初始参考点开始一直连续提供服务的能力，它通常用平均无故障时间来衡量。

可用性——指系统正常工作的时间在连续两次正常服务间隔时间中所占的比率。

可信性——指服务的质量，即在多大程度上可以合理地认为服务是可靠的。

RAID——廉价磁盘冗余阵列或独立磁盘冗余阵列。

分离事务总线——将总线事务分成请求和应答两部分。在请求和应答之间的空闲时间内，总线可以供给其他的 I/O 使用。采用这种技术的总线称为分离事务总线。

通道——专门负责整个计算机系统输入/输出工作的专用处理机，能执行有限的一组输入输出指令。

通道流量——指一个通道在数据传送期间，单位时间内能够传送的数据量。

虚拟 DMA——它允许 DMA 设备直接使用虚拟地址，并在 DMA 传送的过程中由硬件将虚拟地址转换为物理地址。

2. 选择题

【题 8.2】　答：C

【题 8.3】　答：A

【题 8.4】　答：B

【题 8.5】　答：C

【题 8.6】　答：B

3. 填空题

【题 8.7】　答：I/O 设备、I/O 设备与处理机的连接

【题 8.8】　答：连接特性、I/O 系统的容量、响应时间、吞吐率

【题 8.9】　答：平均修复时间、平均无故障时间

【题8.10】 答：细粒度、粗粒度、较小、较大

【题8.11】 答：软件方式、阵列卡方式、子系统方式

【题8.12】 答：CPU-存储器、I/O

【题8.13】 答：同步总线、异步总线

【题8.14】 答：存储器、Cache

【题8.15】 答：编制通道程序、进行正常结束的登记等工作或进行异常处理

【题8.16】 答：寄存器部分、控制部分

【题8.17】 答：字节多路通道、选择通道、数组多路通道

【题8.18】 答：连接在这个通道上的所有设备的数据传输率之和、连接在这个通道上的所有设备中数据流量最大的那一个

【题8.19】 答：访管、管理程序、通道程序、启动I/O

【题8.20】 答：$(T_S+T_D)pn$、$\left(\frac{T_S}{n}+T_D\right)pn$、$\left(\frac{T_S}{k}+T_D\right)pn$

【题8.21】 答：最大、实际

4. 问答题

【题8.22】 答：提高系统组成部件的可靠性的方法：有效构建方法、纠错方法。

有效构建是指在构建系统的过程中消除故障隐患，这样建立起来的系统就不会出现故障。

纠错方法是指在系统构建中采用容错的方法。这样即使出现故障，也可以通过容错信息保证系统正常工作。

【题8.23】 答：①RAID0。也称数据分块，即把数据分布在多个盘上，实际上是非冗余阵列，无冗余信息。②RAID1。也称镜像盘，使用双备份磁盘。每当数据写入一个磁盘时，将该数据也写到另一个冗余盘，这样形成信息的两份复制品。如果一个磁盘失效，系统可以到镜像盘中获得所需要的信息。镜像是最昂贵的解决方法。特点是系统可靠性很高，但效率很低。③RAID2。位交叉式海明编码阵列。原理上比较优越，但冗余信息的开销太大，因此未被广泛应用。④RAID3。位交叉奇偶校验盘阵列，是单盘容错并行传输的阵列。即数据以位或字节交叉的方式存于各盘，冗余的奇偶校验信息存储在一台专用盘上。⑤RAID4。专用奇偶校验独立存取盘阵列。即数据以块(块大小可变)交叉的方式存于各盘，冗余的奇偶校验信息存在一台专用盘上。⑥RAID5。块交叉分布式奇偶校验盘阵列，是旋转奇偶校验独立存取的阵列。即数据以块交叉的方式存于各盘，但无专用的校验盘，而是把冗余的奇偶校验信息均匀地分布在所有磁盘上。⑦RAID6。双维奇偶校验独立存取盘阵列。即数据以块(块大小可变)交叉的方式存于各盘，冗余的检、纠错信息均匀地分布在所有磁盘上。并且，每次写入数据都要访问一个数据盘和两个校验盘，可容忍双盘出错。

【题8.24】 答：

(1) 数据交叉存放的粒度；

(2) 冗余数据的计算方法以及在磁盘阵列中的存放方式。

【题8.25】 答：①同步总线。同步总线上所有设备通过统一的总线系统时钟进行同步。同步总线成本低，因为它不需要设备之间互相确定时序的逻辑。但是同步总线也有缺

点，总线操作必须以相同的速度运行。②异步总线。异步总线上的设备之间没有统一的系统时钟，设备自己内部定时。设备之间的信息传送用总线发送器和接收器控制。异步总线容易适应更广泛的设备类型，扩充总线时不用担心时钟时序和时钟同步问题。但在传输时，异步总线需要额外的同步开销。

【题 8.26】 答：①接收 CPU 发来的 I/O 指令，根据指令要求选择一台指定的外围设备与通道相连接。②执行 CPU 为通道组织的通道程序，从主存中取出通道指令，对通道指令进行译码，并根据需要向被选中的设备控制器发出各种操作命令。③给出外围设备的有关地址，即进行读/写操作的数据所在的位置。④给出主存缓冲区的首地址，这个缓冲区用来暂时存放从外围设备上输入的数据，或者暂时存放将要输出到外围设备的数据。⑤控制外围设备与主存缓冲区之间数据交换的个数，对交换的数据个数进行计数，并判断数据传送工作是否结束。⑥指定传送工作结束时要进行的操作。⑦检查外围设备的工作状态是正常或故障。根据需要将设备的状态信息送往主存指定单元保存。⑧在数据传输过程中完成必要的格式变换。

【题 8.27】 答：①在用户程序中使用访管指令进入管理程序，由 CPU 通过管理程序组织一个通道程序，并启动通道。②通道处理机执行 CPU 为它组织的通道程序，完成指定的数据 I/O 工作。③通道程序结束后向 CPU 发中断请求。CPU 响应这个中断请求后，第二次进入操作系统，调用管理程序对 I/O 中断请求进行处理。

【题 8.28】 答：①字节多路通道。一种简单的共享通道，主要为多台低速或中速的外围设备服务。②数组多路通道。适于为高速设备服务。③选择通道。为高速外围设备（如磁盘存储器等）服务的。

【题 8.29】 答：字节通道每连接一台外设，只传送一个字节，然后又选择与另一台设备连接，并传送一个字节，如此反复。以分时方式轮流为多台低速外设服务。

选择通道在一段时间内只能单独为一台高速外设服务，当这台设备的数据传送工作全部完成后，通道才能为另一台设备服务。选择通道实际上是逐个为物理上连接的几台高速外设服务的。

数组多路通道每连接一台高速设备后，就连续传送一个 k 个字节的数据块，传送完成后，又与另一台高速设备连接，同样连续传送一个 k 个字节的数据块。

【题 8.30】 答：(1)存储器中可能不是 CPU 产生的最新数据，所以 I/O 系统从存储器中取出来的是陈旧数据。

(2) I/O 系统与存储器交换数据之后，在 Cache 中，被 CPU 使用的可能就会是陈旧数据。

第一个问题可以用写直达 Cache 解决。

对于第二个问题，操作系统可以保证 I/O 操作的数据不在 Cache 中。如果不能保证，就作废 Cache 中相应的数据。

5. 应用题

【题 8.31】

解：本题要求计算通道的吞吐率，而且机器有一个多路通道，这就有两种可能：字节多路通道和数组多路通道。因为如果将多路通道组织成数组多路通道，某个时刻通道只能为

一台设备传送数据，所以它的传输率是所有设备的传输率的最大值，而如果将它组织成字节多路通道，该通道的最大传输率就是所有设备的传输率之和。

所以在本题中，从性能上考虑，应组织成字节多路通道形式。

所以此类通道的最大传输率为：

(1) $f_{BYTE} = \sum f_i = f_{打印机传输率} \times 2 + f_{读卡机传输率} \times 2 + f_{终端传输率} \times 10 = 25.6KB/s$ $(i = 1 \sim 14)$

(2) 两个选择通道连接的设备相同，所以只要计算其中一个通道的传输率既可。因为磁盘机的传输率大于磁带机。所以此类通道的传输率为：

$$\max\{800,200\}=800KB/s$$

所以本系统的最大数据传输率为：$f_{系统}=2\times800+25.6=1625.6KB/s$。

【题8.32】

解：数组多路通道的最大流量为：

$$f_{max} = \frac{1}{\frac{T_S}{k}+T_D} = \frac{k}{T_S+kT_D} = \frac{512}{1+512\times1} = \frac{512}{513}(B/\mu s) < 1000KB/s$$

连接到通道上的外设能够正常工作的条件是该外设的数据传输速率 f_i 满足：

$$f_i \leqslant f_{max}$$

因此，可以连接到通道上正常工作的外设为：D_2、D_3、D_4、D_5、D_6。

【题8.33】

解：(1) 如果按字节多路通道设计，该通道的最大流量为：

$$f_{max\text{-}byte} = \frac{1}{T_S+T_D} = 250KB/s$$

连接到字节多路通道上的 p 台设备能正常工作的条件是：$f_{max\text{-}byte} \geqslant \sum_{i=1}^{p} f_i$

应选择外设3、4、5、7和8同时连接到通道上，因为

$$\sum_{i=1}^{5} f_i = 100+75+50+14+10 = 249KB/s < 250KB/s$$

(2) 如果按字节多路通道设计，该通道的最大流量为：

$$f_{man=block} = \frac{k}{T_S+kT_D} = \frac{512}{2+512\times2} = 0.499B/\mu s = 499KB/s$$

连接到数组多路通道上的设备能正常工作的条件是：$f_{max\text{-}block} \geqslant f_i$

因此，除外设1外，其他外设都可以同时连接到通道上。

【题8.34】

解：字节多路通道的最大流量为：$f_{max\text{-}byte} = \dfrac{1}{T_S+T_D}$

字节多路通道的实际流量为：$f_{byte} = \sum_{i=1}^{p} f_i$

其中，p 为通道连接的外设台数，f_i 为外设 i 的数据传输速率。因为连接的是同样的外设，所以 $f_1=f_2=\cdots=f_p=f$，故有 $f_{byte}=pf$。

通道流量匹配的要求有：$f_{max\text{-}byte} \geqslant f_{byte}$

即有：$\frac{1}{T_S+T_D}\geqslant pf$；可得：$p\leqslant\frac{1}{(T_S+T_D)f}$

已知 $T_S=9.8\mu s$，$T_D=0.2\mu s$，$1/f=500\mu s$，可求出通道最多可连接的设备台数为：

$$p\leqslant\frac{1}{(T_S+T_D)f}=\frac{500\mu s}{(9.8+0.2)\mu s}=50$$

【题 8.35】

解：(1) 通道实际流量为：

$$f_{\text{byte}}=\sum_{i=1}^{6}f_i=50+50+40+25+25+10=200\text{B/ms}$$

(2) 由于通道的最大流量等于实际工作流量，即有

$$f_{\text{max-byte}}=\frac{1}{T_S+T_D}=200\text{B/ms}$$

可得，通道的工作周期 $T_S+T_D=5\mu s$。

【题 8.36】

解：(1) 字节多路子通道 A1 和 A2 的实际流量分别为：

$$f_{\text{byte-A1}}=\sum_{i=1}^{8}f_i=64+56+56+20+20+20+10+10=256\text{KB/s}$$

$$f_{\text{byte-A2}}=\sum_{i=1}^{8}f_i=56+30+30+30+30+30+30+20=256\text{KB/s}$$

字节多路通道 A 的实际流量为：$f_{\text{byte-A}}=f_{\text{byte-A1}}+f_{\text{byte-A2}}=512\text{KB/s}$

数组多路通道 B1 和 B2 的实际流量分别为：

$$f_{\text{block-B1}}=\max_{i=1}^{5}\{f_i\}=\max\{512,512,512,256,256\}=512\text{KB/s}$$

$$f_{\text{block-B2}}=\max_{i=1}^{5}\{f_i\}=\max\{256,256,512,256,512\}=512\text{KB/s}$$

选择通道 C 的实际流量为：

$$f_{\text{select-C}}=\max_{i=1}^{5}\{f_i\}=\max\{1024,512,512,512,512\}=1024\text{KB/s}$$

通道的最大流量应大于或等于通道实际流量，通道才能正常工作，因此，各通道的最大流量应为：

$$f_{\text{max-byte-A}}\geqslant 512\text{KB/s}$$
$$f_{\text{max-block-B1}}\geqslant 512\text{KB/s}$$
$$f_{\text{max-block-B2}}\geqslant 512\text{KB/s}$$
$$f_{\text{max-select-C}}\geqslant 1024\text{KB/s}$$

(2) I/O 系统的实际流量 $f_{I/O}$ 为各通道实际流量之和，即

$$f_{I/O}=f_{\text{byte-A}}+f_{\text{block-B1}}+f_{\text{block-B2}}+f_{\text{select-C}}=(512+512+512+1024)\text{KB/s}$$
$$=2560\text{KB/s}=2.5\text{MB/s}$$

若 I/O 系统实际流量为主存带宽的 1/2，那么，主存带宽 f_M 应达到

$$f_M=2f_{I/O}=5\text{MB/s}$$

【题 8.37】

解：每个主存页有 32K/128=256 块。

因为是按块传输,所以 I/O 传输本身并不引起 Cache 不命中。但是它可能要替换 Cache 中的有效块。如果这些被替换块中有 60%是被修改过的,将需要(256×60%)×30=4608 个时钟周期将这些被修改过的块写回主存。

这些被替换出去的块中,有 95%的后继需要访问,从而产生 95%×256=244 次不命中,将再次发生替换。由于这次被替换的 244 块中数据是从 I/O 直接写入 Cache 的,因此所有块都为被修改块,需要写回主存(因为 CPU 不会直接访问从 I/O 来的新页中的数据,所以它们不会立即从主存中调入 Cache),需要时间是 244×(40+30)=17 080 个时钟周期。

没有 I/O 时,每一页平均使用 200 万个时钟周期,Cache 不命中 36 000 次,其中 60%被修改过,所需的处理时间为:

$$(36\,000\times 40\%)\times 40+(36\,000\times 60\%)\times(40+30)=2\,088\,000(\text{时钟周期})$$

时钟 I/O 造成的额外性能损失比例为:

$$(4608+17\,080)\div(2\,000\,000+2\,088\,000)=0.53\%$$

即大约产生 0.53%的性能损失。

第9章 互连网络

9.1 基本要求与难点

9.1.1 基本要求

(1) 掌握有关互连网络的基本概念。

(2) 掌握几种常用的基本互连函数及其主要特点。

(3) 掌握互连网络的结构参数与性能指标。

(4) 掌握线性阵列、环等9种静态互连网络的结构及特征。

(5) 理解总线网络、交叉开关网络和多级互连网络3种动态互连网络，掌握其各自的主要特点。

(6) 深入掌握多级立方体网络和Omega网络。

(7) 掌握4种消息寻径方式：线路交换，存储转发，虚拟直通和虫蚀方式。

(8) 了解流控制策略。

(9) 了解选播和广播寻径算法。

9.1.2 难点

(1) 静态互连网络的结构及特征。

(2) 多级立方体网络和Omega网络。

(3) 消息寻径方式。

9.2 知识要点

互连网络是一种由开关元件按照一定的拓扑结构和控制方式构成的网络，用来实现计算机系统中结点之间的相互连接。这些结点可以是处理器、存储模块或其他设备。在拓扑上，互连网络是输入结点到输出结点之间的一组互连或映像。

互连网络有三大要素：互连结构，开关元件，控制方式。各种互连网络主要是在这些要素方面各不相同。

9.2.1 互连函数

假设互连网络有 N 个输入端和 N 个输出端，分别用 $0,1,\cdots,N-1$ 来表示，则互连函数表示了输入端号和输出端号的连接关系。

1. 互连函数的表示方法

用变量 x 表示输入(设 $x=0,1,\cdots,N-1$)，用函数 $f(x)$ 表示输出。则 $f(x)$ 表示：在互连函数 f 的作用下，输入端 x 连接到输出端 $f(x)$。x 和 $f(x)$ 可以用二进制表示，也可以用十进制表示。

互连函数反映了网络输入端数组和输出端数组之间对应的置换关系或排列关系，所以互连函数有时也称为**置换函数**或**排列函数**。

互连函数的**循环表示法**：$(x_0\ x_1\ x_2 \cdots\ x_{j-1})$

它表示：$f(x_0)=x_1, f(x_1)=x_2, \cdots, f(x_{j-1})=x_0$

j 称为该循环的长度。

2. 几种基本的互连函数

下面介绍几种常用的基本互连函数及其主要特征。

1) 恒等函数

恒等函数所实现的互连：同号输入端和输出端之间的连接。

其函数表达式为：

$$I(x_{n-1}x_{n-2}\cdots x_1x_0) = x_{n-1}x_{n-2}\cdots x_1x_0$$

2) 交换函数

交换函数所实现的互连：二进制地址编码中第 k 位互反的输入端与输出端之间的连接。其表达式为：

$$E(x_{n-1}x_{n-2}\cdots x_{k+1}x_kx_{k-1}\cdots x_1x_0) = x_{n-1}x_{n-2}\cdots x_{k+1}\ \bar{x}_kx_{k-1}\cdots x_1x_0$$

3) 均匀洗牌函数

均匀洗牌函数：将输入端分成数目相等的两半，前一半和后一半按类似均匀混洗扑克牌的方式交叉地连接到输出端(输出端相当于混洗的结果)。其函数关系可表示为：

$$\sigma(x_{n-1}x_{n-2}\cdots x_1x_0) = x_{n-2}x_{n-3}\cdots x_1x_0x_{n-1}$$

即把输入端的二进制编号循环左移一位，就变成了其所要连接的输出端的编号。

对于有些互连函数(设为 s)，还可以定义其**第 k 个子函数**和**第 k 个超函数**。它们分别是把 s 作用于输入端的二进制编号的低 k 位和高 k 位。

对于任意一种函数 $f(x)$，如果存在 $g(x)$，使得

$$f(x) \times g(x) = I(x)$$

则称 $g(x)$ 是 $f(x)$ 的**逆函数**，记为 $f^{-1}(x)$。即 $f^{-1}(x)=g(x)$。

逆均匀洗牌是均匀洗牌的逆函数，记为 σ^{-1}。它所实现的函数是：

$$\sigma^{-1}(x_{n-1}\ x_{n-2}\cdots\ x_1x_0) = x_0x_{n-1}x_{n-2}\cdots x_1$$

即把输入端的二进制编号循环右移一位。

逆均匀洗牌和均匀洗牌的连接图互为镜像。均匀洗牌和逆均匀洗牌是两种十分有用的互连函数，用它们和交换开关多级组合起来可构成 Omega 网络和逆 Omega 网络。

4）碟式函数

蝶式互连函数定义为：

$$\beta(x_{n-1}x_{n-2}\cdots x_1x_0)=x_0x_{n-2}\cdots x_1x_{n-1}$$

即把输入端的二进制编号的最高位与最低位互换位置，就得到了其所要连接的输出端的编号。

与均匀混洗函数类似，只用蝶式函数不能实现任意结点之间的连接。但是蝶式变换与交换变换的多级组合可作为构成方体多级网络的基础。

5）反位序函数

反位序函数的定义为：

$$\rho(x_{n-1}x_{n-2}\cdots x_1x_0)=x_0x_1\cdots x_{n-2}x_{n-1}$$

即把输入端二进制编号的各位的次序颠倒过来，便得到了其所连输出端的编号。

6）移数函数

移数函数是将各输入端都错开一定的位置（模 N）后连到输出端。其函数式为：

$$\alpha(x)=(x\pm k)\bmod N\quad 1\leqslant x\leqslant N-1,1\leqslant k\leqslant N-1$$

还可以将整个输入数组（把输入端号按顺序排列）分成若干个子数组，然后在子数组的范围内进行移数置换。

7）PM2I 函数

PM2I 函数中的 P 和 M 分别表示加（Plus）和减（Minus），2I 表示 2^i，所以该函数又称为"加减 2^i"函数。这是一种特殊的移数函数，其函数式为：

$$\mathrm{PM2}_{+i}(x)=(x+2^i)\bmod N$$

$$\mathrm{PM2}_{-i}(x)=(x-2^i)\bmod N$$

其中，$0\leqslant x\leqslant N-1$，$0\leqslant i\leqslant n-1$，$n=\log_2 N$，$N$ 为结点数。显然，PM2I 互连网络共有 $2n$ 个互连函数。

PM2I 函数是构成数据变换网络的基础。阵列计算机 Illiac Ⅳ 采用 $\mathrm{PM2}_{\pm 0}$ 和 $\mathrm{PM2}_{\pm n/2}$ 构成其互连网络，实现各处理单元之间的上下左右互连。

9.2.2　互连网络的结构参数与性能指标

1. 互连网络的结构参数

互连网络通常是用有向边或无向边连接有限个结点的图来表示。其主要结构参数包含以下 6 个。

（1）网络规模 N。网络规模是指互连网络中结点的个数。它表示该网络所能连接的部件的数量。网络规模越大，则这个互连网络的连接能力越强，能连接更多的部件。

（2）结点度 d。结点度 d 是指互连网络中结点所连接的边数（通道数），包括入度和出度。进入结点的边数叫入度，从结点出来的边数叫出度。

（3）结点距离。对于互连网络中的任意两个结点，其距离是指从一个结点出发到另一

个结点终止所需要跨越的边数的最小值。

(4) 网络直径 D。网络直径是指互连网络中任意两个结点之间距离的最大值。网络中任意两个结点之间传送信息所通过的边数都不会大于网络直径。显然，网络直径应当尽可能地小。

(5) 等分宽度 b。把由 N 个结点构成的网络切成结点数相同($N/2$)的两半，在各种切法中，沿切口边数的最小值称为该网络的等分宽度，用 b 表示。而线等分宽度为 $B=b\times w$。其中，w 为通道宽度(用位表示)。该参数主要反映了网络的最大流量。

(6) 对称性。如果从任意结点来看，网络的结构都是相同的，则称该网络为对称网络。对称网络实现比较容易，编程也比较容易。

2. 互连网络的性能指标

时延和带宽是用来评估互连网络性能的两个基本指标。

1) 通信时延

通信时延是指从源结点到目的结点传送一条消息所需的总时间，它由以下4部分构成。

(1) 软件开销：在源结点和目的结点用于收发消息的软件所需的执行时间。

(2) 通道时延：通过通道传送消息所花的时间。通道时延=消息长度/通道带宽。

(3) 选路时延：消息在传送路径上所需的一系列选路决策所需的时间开销。

(4) 竞争时延：多个消息同时在网络中传送时，会发生争用网络资源的冲突。为避免或解决争用冲突所需的时间。

软件开销主要取决于两端结点处理消息的软件内核。通道时延通常由瓶颈链路的通道带宽决定。选路时延与传送路径上的结点数成正比。竞争时延很难预测，它取决于网络的传输状态。

2) 网络时延

网络时延是指通道时延与选路时延的和。它是由网络硬件特征决定的，与程序行为和网络传输状态无关。而软件开销和竞争时延则是与程序的行为有关。

3) 端口带宽

对于互连网络中的任意一个端口来说，其端口带宽是指单位时间内从该端口传送到其他端口的最大信息量。

在对称网络中，端口带宽与端口位置无关。网络的端口带宽与各端口的端口带宽相同。而非对称网络的端口带宽则是指所有端口带宽的最小值。

4) 聚集带宽

网络的聚集带宽是指网络从一半结点到另一半结点，单位时间内能够传送的最大信息量。例如，HPS是一种对称网络，网络规模 N 的上限是512，端口带宽是40MB/s，因此，HPS的聚集带宽是：(40MB/s×512)/2=10.24GB/s。

5) 等分带宽

与等分宽度对应的切平面中，所有边合起来单位时间内所能传送的最大信息量称为该网络的等分带宽。

9.2.3　静态互连网络

互连网络通常可以分为两大类：静态互连网络和动态互连网络。**静态互连网络**是指各结点之间有固定的连接通路、且在运行中不能改变的网络。而**动态互连网络**则是指由交换开关构成、可按运行程序的要求动态地改变连接状态的网络。

下面介绍几种静态互连网络，其中的 N 表示网络的规模，即结点的个数。

1. 线性阵列

线性阵列是最简单的拓扑结构，其中 N 个结点用 $N-1$ 条链路连成一行。其优点是实现成本低。但这种结构不对称，而且当 N 较大时，直径也比较大，通信效率低。

线性阵列与总线的区别是很大的。

2. 环和带弦环

用一条链路将线性阵列的两个端结点连接起来即可得到环。环可以单向工作，也可以双向工作。它是对称的。

在环的基础上，给每个结点增加一条或两条链路，即可得到两种带弦环。增加的链路愈多，结点度愈高，网络直径就愈小。极端情况：全连接网络，结点度为 $N-1$，直径为 1。

3. 循环移数网络

循环移数网络是通过在环上增加以下链路而构成的：每个结点到所有与其距离为 2 的整数幂的结点之间都增加一条链路。也就是说，如果 $|j-i|=2^r$，$r=0,1,2,\cdots,n-1$，($n=\log_2 N$)，则结点 i 与结点 j 连接。

显然，与结点度较低的任何带弦环相比，循环移数网络都具有更好的连接特性。对 $N=16$ 的情况，循环移数网络的结点度为 7，直径为 2。它的复杂性比全连接网络低得多。

4. 树状和星状

一般来说，一棵 k 层完全平衡的**二叉树**有 $N=2^k-1$ 个结点。其最大的结点度 $d=3$，由于结点度是常数，因此二叉树是一种可扩展的结构。其缺点是直径比较长。

星状网络是一种两层的树，结点度为 $d=N-1$，直径 D 很小，为 2。但其可靠性比较差，只要中心结点出故障，整个系统就会瘫痪。

5. 胖树状

树状网络的一个主要缺点是其根部结点以及连到根部的链路上的负载比较重，有可能会成为整个系统的瓶颈。胖树的提出使该问题得到了缓解。在二叉胖树中，胖树的通道宽度从叶结点往根结点方向逐渐增宽，它更像真实的树，愈靠近树根，树干就愈粗。

6. 网格状和环状

网格状是一种比较流行的结构，它已经以各种变体形式在 Illiac Ⅳ、CM-2 和 Intel

Paragon等机器中得到了实现。

一般来说，一个由 $N=n^k$ 个结点构成的 k 维网格状网络(每维 n 个结点)的内部结点度 $d=2k$，网络直径 $D=k(n-1)$。

Illiac 网络的名称来源于采用了这种网络的 Illiac Ⅳ 计算机。它是二维网格状网络的一种变形。把二维网格状网络的每一列的两个端结点连接起来，再把每一行的尾结点与下一行的头结点连接起来，并把最后一行的尾结点与第一行的头结点连接起来，就形成了**Illiac 网络**。

把二维网格状网络的每一行的两个端结点连接起来，把每一列的两个端结点也连接起来，就成了**环状网络**。环网是一种对称的拓扑结构，所有附加的回绕连接使得原来网格结构的直径减少了一半。这种拓扑结构将环状和网格状组合在一起，并能向高维扩展。

7. 超立方体

这是一种二元 n 立方体结构，一般来说，一个二元 n 立方体由 $N=2^n$ 个结点组成，它们分布在 n 维上，每维有两个结点。

将两个3-立方体中相对应的结点用链路连接起来，便可形成4-立方体。以此类推，便可形成更多维的立方体。一般地，如果要形成一个规模为 $N=2^n$ 的 n 立方体，只要把两个 $(n-1)$-立方体中相对应的结点用链路连接起来即可，共需要连接 2^{n-1} 条链路。由于对于 n 立方体中的每一个结点来说，在其每一维的方向上，都有且仅有一个结点与其相连，所以结点的度 $d=n$。n 维立方体的直径 $D=n$，等分宽度为 $b=N/2$。超立方体结构的扩展十分困难。超立方体的网络结构是对称的。

8. 带环立方体

一般来说，带环 k-立方体(简称 k-CCC)是 k-立方体的变形，它是通过用 k 个结点构成的环取代 k-立方体中的每个结点而形成的。

前面讲过，k 维立方体中每个结点的度都是 k，即有 k 条边与之相连。当用由 k 个结点构成的环取代这个结点(设为 a)时，让该环中的各结点分别与 a 所连接的 k 条边连接。这样一来，不管带环 k-立方体的 k 有多大，所有结点的度都是常数3，与超立方体的维数无关，因而具有很好的扩展性。

9. k 元 n-立方体网络

环状、网状、环网状、二元 n 立方体(超立方体)和 Omega 网络都是 k 元 n 立方体网络系列的拓扑同构体。在 k 元 n 立方体网络中，参数 n 是立方体的维数，k 是基数，即每一维上的结点个数。这两者的关系是：

$$N=k^n,\quad (k=\sqrt[n]{N},n=\log_k N)$$

按照惯例，通常把低维 k 元 n-立方体称为环网，而把高维 k 元 n-立方体称为超立方体。

大多数网络的结点度都不超过4，这是比较理想的。全连接网络和星状网络的结点度太高。超立方体的结点度随 $\log_2 N$ 值的增大而增大，当 N 值很大时其结点度也太高。

网络直径的变化范围很大。虽然直径小仍然是一种优点，但随着硬件寻径技术不断革新，直径已不是一个严重的问题，因为在采用高度流水技术后，任意两结点间的通信延迟几

乎是固定不变的。链路数会影响网络价格，等分宽度将影响网络的带宽，可以通过采用较宽的通道来扩大等分宽度。

对称性会影响可扩放性和寻径效率。网络的总价格随网络直径和链路数的增大而上升。直径小仍然是一个优点，但结点之间的平均距离可能是一种更好的度量指标。

9.2.4 动态互连网络

动态互连网络设置有源开关，因而能够根据需要借助控制信号对连接通路加以重新组合，实现所要求的通信模式。下面介绍总线网络、交叉开关网络和多级互连网络。

1. 总线网络

总线由一组导线和插座构成，经常被用来实现计算机系统中处理机模块、存储模块和外围设备等之间的互连。总线系统具有结构简单、实现成本低等优点。但由于是由许多模块分时共享，每次只能处理一个请求，所以它很容易就会成为系统的瓶颈。

为了解决总线带宽较窄的问题，可以采用多总线或多层次的总线。

2. 交叉开关网络

可以把**交叉开关网络**看作一个单级开关网络。像电话交换机一样，交叉点开关能在对偶（源、目的）之间形成动态连接，同时实现多个对偶之间的无阻塞连接。开关可根据程序的要求动态地设置为“开”或“关”。与其他的动态互连网络相比，交叉开关网络的带宽和互连特性是最好的。对于一个 $n \times n$ 的交叉开关网络，可以无阻塞地实现 $n!$ 种置换。不过，对一个 $n \times n$ 的交叉开关网络来说，需要 n^2 套交叉点开关以及大量的连线。当 n 很大时，交叉开关网络所需要的硬件数量非常巨大。因此，一般只有 $n \leqslant 16$ 的小型交叉开关网络用在商品化的机器中。

我们可以在处理机和存储器模块之间用交叉开关网络互连，构成一个共享存储器的多处理机。这里，每个存储模块一次只能满足一台处理机的请求。当多个请求要同时访问同一存储模块时，交叉开关就必须分解所发生的冲突，每一列只能接通一个交叉点开关。但是，每一台处理机可能会产生一系列地址要同时访问多个存储模块。为了支持并行（或交叉）存储器访问，可以在同一行中接通几个交叉点开关。

处理机之间的交叉开关可以实现处理机之间的置换连接，但这只是一对一的连接。所以 $n \times n$ 交叉开关网络一次最多可实现 n 个“源-目的”对的连接。

3. 多级互连网络

1）多级互连网络的构成

MIMD 和 SIMD 计算机都使用多级互连网络 MIN。各种多级互连网络的区别在于所用开关模块、控制方式和级间互连模式的不同。

常用的开关模块为：2×2、4×4、8×8。这些模块中，每个输入可与一个或多个输出相连，但是在输出端不许发生冲突。最简单的开关模块是 2×2 开关。可用作各种多级互连网络的基本构件。2×2 开关有 4 种连接方式：直送，交叉，上播，下播。

控制方式是指对各个开关模块进行控制的方式,有以下3种选择。

(1) 级控制:每一级的所有开关只用一个控制信号进行控制,这些开关只能同时处于同一种状态。

(2) 单元控制:每一个开关都有一个独立的控制信号,可各自处于不同的状态。

(3) 部分级控制:第 i 级的所有开关分别用 $i+1$ 个信号控制,$0 \leq i \leq n-1$,n 为级数。

常用的级间互连模式包括均匀洗牌、蝶式、多路洗牌、纵横交叉、立方体连接等。

2) 多级立方体网络

多级立方体网络包括**STARAN 网络**和**间接二进制 n 方体网络**等。这两者仅在控制方式上不同,在其他方面都是一样的。它们都是采用二功能(直送和交换)的 2×2 开关。当第 i 级($0 \leq i \leq n-1$)交换开关处于交换状态时,实现的是 Cube_i 互连函数。

STARAN 网络采用级控制和部分级控制,而间接二进制 n 方体网络则采用单元控制。因而后者具有更大的灵活性。当 STARAN 网络采用级控制时,所实现的是交换功能;而采用部分级控制时,则能实现移数功能。

所谓**交换**,是指将有序的一组元素头尾对称地进行交换。

3) Omega 网络

一般来说,一个 N 输入的 Omega 网络有 $\log_2 N$ 级,每级用 $N/2$ 个四功能的 2×2 开关模块,共需要 $\log_2 N \times N/2$ 个开关。每个开关模块均采用单元控制方式。

Omega 网络与间接二进制 n 方体网络在拓扑结构上正好是互逆的。所以,如果 Omega 网络也采用二功能交换开关,那么 Omega 网络和间接二进制 n 方体网络就互为逆网络。

4. 动态互连网络的比较

总线互连的复杂性最低,成本也是最低。其缺点是每台处理机可用的带宽较窄。

交叉开关是最昂贵的,因为其硬件复杂性以 n^2 上升,所以其成本最高。但是交叉开关的带宽和寻径性能最好。如果网络的规模较小,它是一种理想的选择。

多级互连网络的复杂度和带宽介于总线和交叉开关之间,是一种折中方案。其主要优点是采用模块化结构,可扩展性较好。不过,其时延随网络级数的增加而上升。另外,由于其硬件复杂度比总线高很多,其成本也不低。

9.2.5 消息传递机制

消息传递机制在实现多处理机或多计算机中各结点之间的互连通信中有重要的作用。在这样的系统中,处理机之间是通过发送消息来进行通信的。当某个处理机(设为 A)要对远程存储器上的数据进行访问(或操作)时,它就通过给相应的远程处理机(设为 B)发送一个消息来请求数据(或对该数据进行操作)。在这种情况下,可以把该消息看成是一个远程进程调用(Remote Process Call,RPC)。当目的处理器 B 接收到消息以后,就代替远程处理器 A 对相应的数据进行访问(或执行相应的操作),然后发送一个应答消息给处理器 A,将结果返回。

当源结点和目的结点之间没有直接的连接时,消息需要经过中间的结点进行传递。**寻径**就是用来实现这种传递的通信方法和算法。有的文献称之为**路由**。

1. 消息寻径方案

1) 消息的格式

消息是结点之间进行通信的逻辑单位。消息一般是由若干个“包”组成。尽管包的长度是固定的，但因一条消息中所包含的包的个数是可变的，所以消息的长度是不定长的。

包是包含寻径所需目的地址的基本单位。由于不同的包可能是异步到达目的结点的，因此一条消息中的各个包都依次被分配一个序号，以便这些包到达目的结点后能重新组装出消息。

包可以进一步分成一些更小的固定长度的单位，称为“片”。寻径信息和包序列号形成头片，其余的是**数据片**。

包的长度主要是由寻径方案和网络的具体实现所决定的，典型的长度是 64～512 位不等；片的长度经常是受网络大小的影响，例如，一个由 256 个结点构成的网络要求片的长度为 8 位。其他影响包的长度和片的长度的因素还有通道带宽、寻径设计、网络的流量密度等。

2) 4 种寻径方式

消息寻径方式可以分为两大类：线路交换和包交换。包交换又分为存储转发、虚拟直通、虫蚀 3 种方式。

(1) 线路交换

在线路交换方式下，在传递一个信息之前，需要先建立一条从源结点到目的结点的物理通路，然后再传递信息。其传输时延 T 为：

$$T = \frac{L + L_t \times (D+1)}{B} \tag{9.1}$$

其中，L 为信息包的长度(位数)，L_t 为建立路径所需的小信息包的长度，D 为经过的中间结点个数，B 为带宽。

线路交换的优点是：传输带宽较大，平均传输时延较小，而且使用的缓冲区小。因而适合于具有动态和突发性的大规模并行处理数据的传送。缺点是：需要频繁地建立源结点到目的结点的物理通路，时间开销会很大。

(2) 存储转发

存储转发是最简单的分组交换方式。在这种方式中，包是信息传递的基本单位。包从源结点经过一系列中间结点到达目的结点。存储转发要求所经过的每个中间结点都要设置一个包缓冲器，用于保存所传递的包。当一个包到达某个中间结点时，该结点先把这个包全部存储起来，然后在出口链路可用而且下一个结点的包缓冲器也可用的情况下，传递给下一个结点。

存储转发方式中，网络的时延与源和目的地之间的距离(跳数)成正比，时延 T_{SF} 为：

$$T_{SF} = \frac{L}{B}(D+1) \tag{9.2}$$

式中，D，L，B 参数的含义同式(9.1)。

存储转发方式的缺点：①包缓冲区大，不利于 VLSI 实现；②网络时延大，与结点距离成正比。

(3) 虚拟直通

虚拟直通是对存储转发方式的一种改进，减少了网络时延。其基本思想是：没有必要等到信息包全部放入缓冲器后再做路由选择，只要接收到用作寻径的包头，就可做出判断。如果结点的输出链路空闲，信息包可以不必存储在该结点的缓冲器中，而是立即传送到下一个结点。如果整条链路都空闲，包就可以立即直达目的结点。这时它就同使用线路交换一样。但是，这种方式还是要求每个结点中有缓冲器。这是因为在输出链路不空闲时，还是要用缓冲器进行存储。所以无论用存储转发还是用虚拟直通，在各中间结点都需要设置缓冲器。

虚拟直通的通信时延为：

$$T=\frac{L+L_h\times(D+1)}{B}\approx\frac{L}{B} \tag{9.3}$$

式中，L_h是信息包寻径头部的长度。

一般来说，$L>>L_h\times(D+1)$，所以 $T\approx L/B$。可以看出，此时通信时延与结点数目无关，这是一个非常大的改进。

当出现寻径阻塞时，虚拟直通方式也只好将整个信息包全部存储在寻径结点中，直到通道不阻塞时才能将信息包发出。这就要求每个结点都有足够大的缓冲区来存储可能出现的最大的信息包。在这一点上，虚拟直通方式与存储转发方式是一样的，同样不利于 VLSI 的实现。

(4) 虫蚀方式

虫蚀方式比虚拟直通又有了改进。它把信息包“切割”成更小的单位——“片”，而且使信息包中各片的传送按流水方式进行，所以不仅可以减少结点中缓冲器的容量，而且还能缩短传送延迟时间。虫蚀方式在新型的多计算机系统中得到了广泛的应用。

虫蚀方式所能处理的最小信息单位是“片”。当一个结点把头片送到下一个结点后，那么接下来就可以把后面的各个片也依次送出。一个结点一旦开始传送一个包中的头片后，这个结点就必须等待这个包的所有片都送出去后，才能传送其他包。不同包的片不能混合在一起传送。

这种虫蚀方式有点儿像虚拟直通，但不同之处在于：当输出通路忙时，结点是把一个片存储到缓冲器中。由于片的大小比包小很多，所以能有效地减少缓冲器的容量，使得它易于用 VLSI 实现。

对于一个受到阻塞的包来说，它的前后连续的各个片可以散布在若干个中间结点中。换句话说，整个包停留在已建立的通路中的某一路径段里。当可以继续向前传送时，头片每向前传送一个结点，其他数据片就跟着相应地向前“蠕动”一步，就好像一条虫那样蠕动前进。

虫蚀方式的通信时延为：

$$T_{WH}=T_f\times D+\frac{L}{B}=\frac{L+L_f\times D}{B}\approx\frac{L}{B} \tag{9.4}$$

式中，L_f是“片”的长度，T_f是片经过一个结点所需时间，$L>>L_f\times D$。从该式看出，通信时延与结点数无关。

虫蚀方式的优点是：①每个结点的缓冲器较小，易于 VLSI 实现；②有较小的网络传输延迟；③通道共享性好，利用率高；④易于实现选播和广播通信模式。

然而，虫蚀方式也有缺点。当消息的一片被阻塞时，整个消息的所有片都将被阻塞在所在结点，占用了结点资源。

2. 死锁与虚拟通道

1）虚拟通道

虚拟通道是两个结点间的逻辑连接，它由源结点的片缓冲区、结点间的物理通道以及接收结点的片缓冲区组成。当物理通道分配给某对缓冲区时，这一对的源缓冲区和接收缓冲区就形成了一条虚拟通道。物理通道是由所有的虚拟通道分时地共享。除了有关的缓冲区和通道以外，还必须用某些通道状态来区分不同的虚拟通道。源缓冲区存放等待使用通道的片。接收缓冲区存放由通道刚刚传送过来的片。通道(电缆或光纤)是它们之间的通信媒介。

虚拟通道也可以用双向通道实现。把两条单向通道组合在一起可以构成一条双向通道。这不仅增加了利用率，还可使通道的频宽加倍。

2）避免死锁

缓冲区或通道上的循环等待会引起死锁。利用虚拟通道方法可以避免一些死锁。但是增加虚拟通道可能会使每个请求可用的有效通道频宽降低。为此，当实现数目很大的虚拟通道时需要用高速的多路选择开关。

3. 流控制策略

当两个或更多的包在某个结点为竞争缓冲区或通道资源发生冲突时，必须使用预先确定好的策略来解决冲突。我们要寻找的是不会引起拥挤或死锁的控制网络流量的策略。

1）包冲突的解决

为了通过通道在两个相邻结点之间传送一个片，要同时具备 3 个条件：①源缓冲区已存有该片；②通道已分配好；③接收缓冲区准备接收该片。

当两个包到达同一个结点时，它们可能都在请求同一个接收缓冲器或者同一个输出通道，这时必须对两个问题进行仲裁：①把通道分配给哪个包？②如何处理被通道拒绝的包？下面给出 4 种解决方案。这里假设是把通道分配给第一个包，第二个包被拒绝。

(1) 把第二个包暂存在缓冲区。

设置两个缓冲区，一个是包缓冲区，另一个是片缓冲区。将第一个包分配给片缓冲区直接送往输出通道，而第二个包则被放入包缓冲区。当以后通道变为可用时，再来传输第二个包。

(2) 阻塞第二个包。即把第一个包送入片缓冲区，同时用门控将第二个包阻塞。不过，并没有把这个包丢弃。

(3) 丢弃第二个包。即把第一个包送入片缓冲区，并把第二个包丢弃。

(4) 绕道。即把第一个包送入片缓冲区，而把第二个包导向其他通道传输。

在上述方法中，丢弃法有可能会造成严重的资源浪费，而且要求重新进行被丢弃包的传输与确认。这种策略现在已很少采用，因为它的包传输率不稳定。绕道法在包寻径方面提供了更多的灵活性，但为了到达目的结点，可能要花费超过实际需要的通道资源，造成浪费。

2）确定性寻径和自适应寻径

常用的寻径方法有两种：确定性寻径和自适应寻径。在**确定性寻径**中，通信路径完全

由源结点地址和目的地址来决定，也就是说，寻径路径是预先唯一地确定好了的，而与网络的状况无关。在自适应寻径方法中，通信的通路每一次都要根据资源或者网络的情况来选择。这样就可以避开拥挤的或者有故障的结点，从而使网络的利用率可以得到改进。

不管采用哪一种寻径，都希望不会产生死锁。

对于一个多维网来说，维序寻径要求对后继通道的选择是按照各维的顺序来进行的。对于二维的网格网络来说，这种寻径方法被称为 **X-Y 寻径**，因为它首先是沿 X 维方向进行寻径，然后再沿 Y 维方向寻找路径。对于超立方体来说，这种寻径方法被称为 **E-cube 寻径**。

对于给定的任意一个源结点 $s=(x_1,y_1)$ 和任意一个目的结点 $d=(x_2,y_2)$，从 s 出发，先沿 X 轴方向前进，直到找到 d 所在的列 x_2；然后再沿 Y 轴方向前进，直到找到目标结点 (x_2,y_2)。

4. 选播和广播寻径算法

多计算机网络中会出现以下 4 种通信模式。

单播：对应于一对一的通信情况，即一个源结点发送消息到一个目的结点。

选播：对应于一到多的通信情况，即一个源结点发送同一消息到多个目的结点。

广播：对应于一到全体的通信情况，即一个源结点发送同一消息到全部结点。

会议：对应于多到多的通信情况。

通道流量和通信时延是常用的两个参数。通道流量可用传输有关消息所使用的通道数来表示。通信时延则用包的最大传输时间来表示。

优化的寻径网络应能以最小流量和最小时延实现相关的通信模式。然而，这两个参数并不是毫不相关的。达到最小流量的同时，并不一定能达到最小时延。相反的情况也是如此。

这与所使用的交换技术有关，在存储转发网络中时延是最重要的问题，而在虫蚀网络中流量对效率的影响则更大。

习　题

1. 概念题

【题 9.1】 解释以下名词

互连网络	互连函数	网络规模	结点度
结点距离	网络直径	等分带宽	对称网络
静态互连网络	动态互连网络	虚拟通道	自适应寻径

2. 选择题

【题 9.2】 逆均匀洗牌函数得到输出端地址的方法是把输入端二进制地址(　　)。

A. 循环左移一位

B. 循环右移一位

C. 从第 k 位开始的低端地址部分循环左移一位

D. 从最高位至第 $n-k-1$ 位的高端地址部分循环左移一位

【题 9.3】 32 个结点的立方体连接的互连函数的个数是(　　)。

A. 6　　B. 5　　C. 4　　D. 3

【题 9.4】 64 个结点的 PM2I 函数的个数是(　　)。

A. 6　　B. 8　　C. 12　　D. 64

【题 9.5】 用循环函数形式表示 8 个结点的 $PM2_{+1}$ 函数，应该是(　　)。

A. (6 4 2 0)(7 5 3 1)　　B. (0 2 4 6)(1 3 5 7)

C. (0 1 2 3 4 5 6 7)　　D. (0 4)(1 5)(2 6)(3 7)

【题 9.6】 Illiac Ⅳ 阵列处理机中，各处理单元之间所用的互连函数是(　　)。

A. $PM2_{\pm 0}$ 和 $PM2_{\pm n/2}$　　B. 立方体 C_0 和 C_1

C. $PM2_{\pm 2}$　　D. 均匀洗牌

【题 9.7】 结构不对称的静态互连网络是(　　)。

A. 线性阵列　　B. 环网　　C. 立方体网络　　D. 全连接网络

【题 9.8】 结构对称的静态互连网络是(　　)。

A. 二叉树　　B. 星状　　C. 二维网格　　D. 超立方体

【题 9.9】 下列静态互连网络中，可扩展性最差的是(　　)。

A. 环网　　B. 网格网　　C. 带环立方体　　D. 立方体

【题 9.10】 多级混洗(均匀洗牌)置换网络又称为(　　)。

A. STARAN 网络　　B. Omega 网络　　C. 交换网络　　D. 移数网络

【题 9.11】 Omega 网络采用(　　)。

A. 2 功能 2×2 开关，单元控制方式　　B. 4 功能 2×2 开关，单元控制方式

C. 2 功能 2×2 开关，级控制方式　　D. 4 功能 2×2 开关，级控制方式

【题 9.12】 三级 STARAN 网络对 8 个输入端实现两组 4 元分组交换后，输入端端号序列[0 1 2 3 4 5 6 7]置换连接的输出端端号序列为(　　)。

A. [7 6 5 4 3 2 1 0]　　B. [3 2 1 0 7 6 5 4]

C. [1 0 3 2 5 4 7 6]　　D. [6 7 4 5 2 3 0 1]

【题 9.13】 下列 4 种消息寻径方式中，不属于包交换的消息寻径方式是(　　)。

A. 存储转发寻径　　B. 虚拟直通寻径　　C. 虫蚀寻径　　D. 线路交换寻径

【题 9.14】 虫蚀寻径以流水方式在各寻径器中顺序传送的是(　　)。

A. 消息　　B. 包　　C. 片　　D. 字节

3. 填空题

【题 9.15】 互连网络有三大要素：________、________和________。

【题 9.16】 在保持网络拓扑结构不变的情况下，网络可扩充的结点数越多，则称网络的________越好。

【题 9.17】 为了反映不同互连网络的连接特性，每种互连网络可用一组________来描述，它表示相互连接的________端号和________端号之间的一一对应关系。

【题 9.18】 设 C 为立方体互连函数，σ 为均匀洗牌函数，β 为蝶式函数，ρ 为反位序函数，则 $C_3(0110)=$________，$\sigma_{(3)}(0110)=$________，$\beta(0110)=$________，$\rho^{(2)}(0110)=$________。

【题9.19】 用互连网络互连16个处理机,编号为0～15。若互连函数为立方体函数$C_2(C_0)$,则7号处理机连至________号处理机。若互连函数为移数函数$PM2_{-3}$,则7号处理机连至________号处理机。

【题9.20】 设C为立方体互连函数,σ为均匀洗牌函数,互连网络的输入端为$X=x_{n-1}x_{n-2}\cdots x_i\cdots x_1x_0$,若该互连网络实现的互连函数为$\sigma(C_i)$,则连接的输出端为________。

【题9.21】 N个结点的PM2I单级网络的最大距离为________。

【题9.22】 N个端的混洗交换网络中,最远的两个入、出端的二进制编号是________和________,其最大距离为$2\log_2 N-1$。

【题9.23】 0～7共8个处理单元经C_2+C_0互连,第7号处理单元将连至第________号处理单元。

【题9.24】 互连网络的结点度是指连接到结点上的________。在有向图中,结点度是________和________之和。

【题9.25】 由N个结点用$N-1$条链路连成的线性阵列,内部结点的度$d=$________,端结点的度$d=$________,直径$D=$________,等分宽度$b=$________。

【题9.26】 各种多级互连网络的区别在于所用________、________和________的不同。

【题9.27】 2×2开关有________、________、________和________4种连接方式。

【题9.28】 一般来说,一个N输入的多级立方体网络有________级,每级用________个2×2开关模块,共需要________个开关。

【题9.29】 在STARAN网络中,采用级控制时,所实现的是________功能;而采用部分级控制时,则能实现________功能。

【题9.30】 SIMD立方体多级互连网络中,第i级的所有开关用$i+1$个控制信号控制,称此为________控制。

【题9.31】 在SIMD互连的多级网络中,实现移数函数功能中最便宜的方案是用________多级控制,且控制方式采用________。

【题9.32】 Illiac Ⅳ 8×8的阵列中,任意两个处理单元之间通信的最短距离不会超过________。

【题9.33】 在消息传递机制中,消息一般是由若干个________组成,包的长度是________,但一条消息中所包含的包的个数是________。包又可以进一步分成一些更小的固定长度的单位,称为________。

【题9.34】 消息寻径方式可以分为________和________两大类。包交换又分为________、________和________3种方式。

【题9.35】 当两个包在传输过程中争用某个中间结点的同一条输出通道时,有4种方法解决包冲突问题。这4种方法是________、________、________和________。

【题9.36】 在互连网络中,如果消息包的寻径路径是预先唯一地确定好了的,与网络的状况无关,那么,这一类寻径方法称为________寻径方法。如果两个消息包在某个结点产生冲突时,被阻塞包可根据目的地址选择其他通道到达目的结点,那么,这一类寻径方法称为________寻径方法。

【题9.37】 评价网络寻径性能的参数是通道流量和通信时延。网络通道流量可用传

输有关消息所使用的________来表示。通信时延则用包的________来表示。

【题 9.38】　多计算机互连网络中会出现 4 种通信模式。一对一的通信模式称为________模式；一对多的通信模式称为________模式；一对全体的通信模式称为________模式；多对多的通信模式称为________模式。

4. 问答题

【题 9.39】　评价互连网络性能的主要参数有哪些？

【题 9.40】　k 元 n- 立方体网络中，参数 k 和 n 的含义是什么？它们与网络中结点数 N 满足什么关系？

【题 9.41】　多级互连网络的控制方式指什么？通常有哪几种？

【题 9.42】　$N=16$ 的 4 级立方体互连网络，级号从输入到输出为 0～3，采用级控制，如将其中的第 1 级开关处于"直连"，不能实现哪些结点之间的配对通信？为什么？

【题 9.43】　试比较可用于动态互连的总线、交叉开关和多级互连网络的硬件复杂度和带宽。

【题 9.44】　什么是线路交换？它有何优缺点？

【题 9.45】　简述存储转发的基本思想。它的主要缺点是什么？

【题 9.46】　简述虫蚀方式的基本思想。它的主要优缺点是什么？

【题 9.47】　在有 16 个处理器的混洗交换网络中，若要使第 0 号处理器与第 15 号处理器相连，需要经过多少次均匀洗牌和交换？

【题 9.48】　列出互连网络中的 4 种信息传递方式，并分别给出其传输时延公式。

5. 应用题

【题 9.49】　设 E 为交换函数，σ 为均匀洗牌函数，β 为蝶式函数，PM2I 为移数函数，函数的自变量是十进制数表示的处理机编号。现有 32 台处理机，其编号为 0，1，2，…，31。

(1) 分别计算下列互连函数。

$E_2(12)$　$\sigma(8)$　$\beta(9)$　$\text{PM2I}_{+3}(28)$　$E_0(\sigma(4))$　$\sigma(E_0(18))$

(2) 用 E_0 和 σ 构成均匀洗牌交换网（每步只能使用 E_0 和 σ 一次），网络直径是多少？从 5 号处理机发送数据到 7 号处理机，最短路径要经过几步？请列出经过的处理机编号。

(3) 采用移数网络构成互连网络，网络直径是多少？结点度是多少？与 2 号处理机距离最近的是几号处理机？

【题 9.50】　设有一个 4 级立方体网络，从网络输入端到输出端的开关级依序为 K_0、K_1、K_2 和 K_3，网络输入端和输出端的编号均为 0～15。对于下述连接，分别写出网络级控制信号和互连函数。

(1) 网络实现的置换连接为 4 组 4 元交换，可以用循环互连函数的形式表示为：

(0　3)(1　2)(4　7)(5　6)(8　11)(9　10)(12　15)(13　14)

(2) 网络实现的置换连接为：4 组 4 元交换＋1 组 16 元交换。

【题 9.51】　在编号分别为 0，1，2，…，F 的 16 个处理器之间，要求按下列配对通信：

(B,1)(8,2)(7,D)(6,C)(E,4)(A,0)(9,3)(5,F)。试选择所用互连网络类型及其控制方式,并画出互连网络的结构,说明各级交换开关的状态。

【题9.52】 16台处理器用Illiac网络互连,写出Illiac网络的互连函数,给出表示任何一个处理器$PU_i(0\leqslant i\leqslant 15)$与其他处理器直接互连的一般表达式。

【题9.53】 $N=16$的互连网络的输入端号和输出端号分别为0~15。若互连网络实现的互连可以用互连函数表示为$f(x_3x_2x_1x_0)=x_0x_1x_2x_3$,那么,是否可以用循环表示法表示出互连网络实现的互连?如果可以,请写出循环表示。

【题9.54】 $N=16$的STARAN网络在级控方式下实现分组交换置换,如果实现的分组交换置换是:首先是4组4元交换,然后是两组8元交换,最后是一组16元交换,写出网络实现的互连函数。

【题9.55】 具有$N=2^n$个输入端的Omega网络,采用单元控制。

(1) N个输入总共应有多少种不同的排列?

(2) 该Omega网络通过一次可以实现的置换总共可有多少种是不同的?

(3) 若$N=8$,计算一次通过能实现的置换数占全部排列的百分比。

【题9.56】 用一个$N=8$的三级Omega网络连接8个处理机($P_0\sim P_7$),8个处理机的输出端分别依序连接Omega网络的8个输入端0~7,8个处理机的输入端分别依序连接Omega网络的8个输出端0~7。如果处理机P_6要把数据播送给处理机$P_0\sim P_4$,处理机P_3要把数据播送给处理机$P_5\sim P_7$,那么,Omega网络能否同时为它们的播送要求实现连接?画出实现播送的Omega网络的开关状态图。

【题9.57】 试证明多级Omega网络采用不同大小构造块构造时所具有的下列特性。

(1) 一个$k\times k$开关模块的合法状态(连接)数目等于k^k。

(2) 试计算用2×2开关模块构造的64个输入端的Omega网络一次通过所能实现置换的百分比。

(3) 采用8×8开关模块构造64个输入端的Omega网络,重复(2)。

(4) 采用8×8开关模块构造512个输入端的Omega网络,重复(2)。

【题9.58】 现有16个处理器,编号分别为0,1,…,15,用一个$N=16$的互连网络互连。处理器i的输出通道连接互连网络的输入端i,处理器i的输入通道连接互连网络的输出端i。当该互连网络实现的互连函数分别为:

(1) Cube_3

(2) PM2_{+3}

(3) PM2_{-0}

(4) σ

(5) $\sigma(\sigma)$

时,分别给出与第13号处理器所连接的处理器号。

【题9.59】 如图9.1所示,输入端为8个处理机,输出端为8个存储器,通过三级立方体互连网络连接,采用级控方式。其中,所有交换开关均为二功能(控制信号为“0”时直通,为“1”时交换)。若级控信号为:①$K_0K_1K_2=100$;②$K_0K_1K_2=110$;③$K_0K_1K_2=111$;请

在表 9.1 中填写出对应于 8 个处理机而实际连通的 8 个存储器的排列顺序。

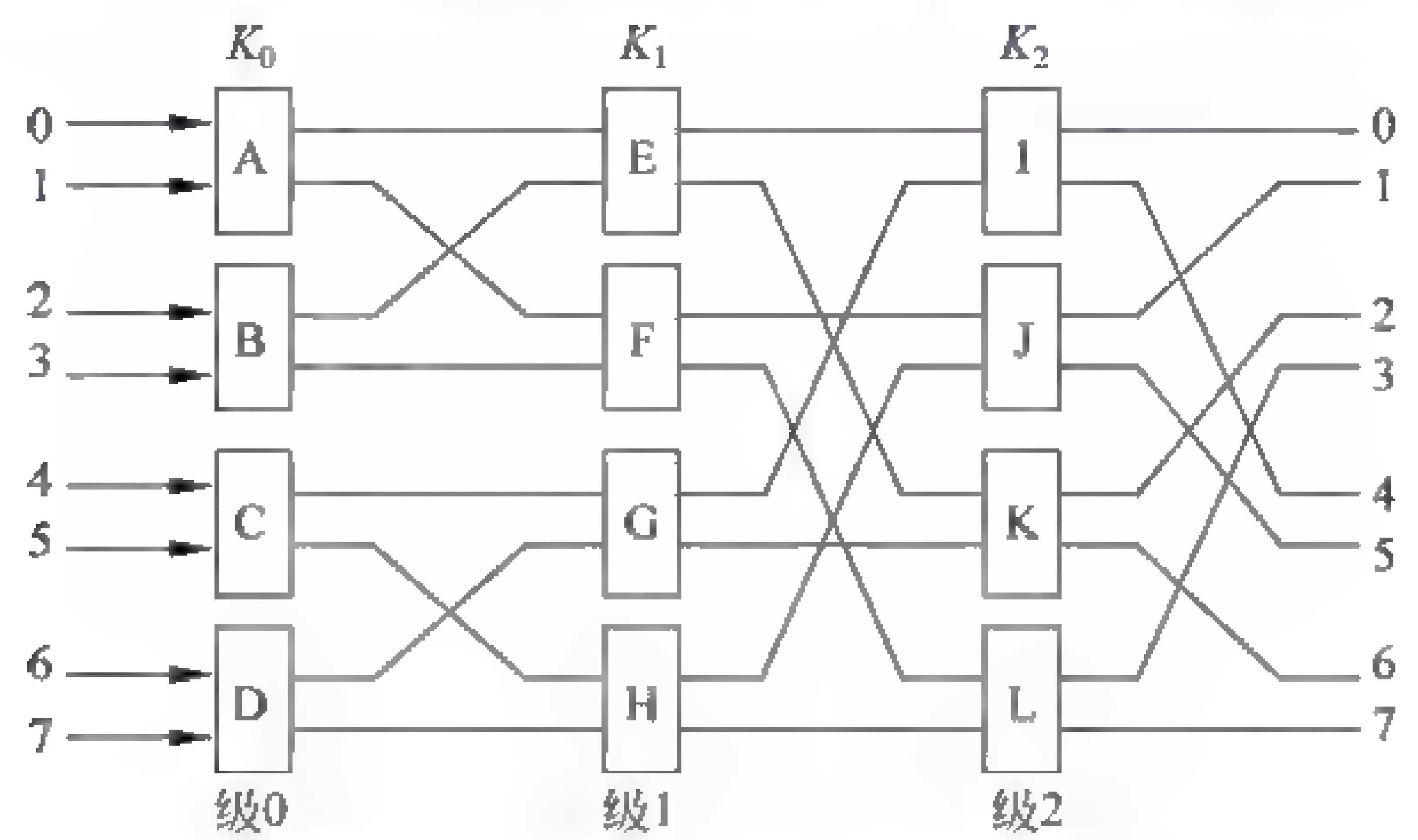


图 9.1　三级立方体互连网络连接

表 9.1　题 9.59

输入排列(处理机)	输出排列(存储器)		
	$K_0K_1K_2=100$	$K_0K_1K_2=110$	$K_0K_1K_2=111$
0			
1			
2			
3			
4			
5			
6			
7			

【题 9.60】 要求用直径最小的三维环网、六维二元超立方体和带环立方体(CCC)设计一台由 64 个结点组成的多计算机直接连接网络。令 d 为网络的结点度，D 为直径，l 为链路总数。假设网络的性能可用$(d\times D\times l)^{-1}$来衡量，试根据其性能列表排出上述 3 种网络的名次。

【题 9.61】 假设循环移数网络有 64 个结点，记为 $N_0,N_1,\cdots,N_{63}$，且网络链路能双向工作。

(1) 列出该网络从结点 N_0 出发，正好以三步可到达的所有结点。

(2) 指出数据从任意一个结点 N_i 传送至另一个结点 N_j 所需的最少寻径步的最小上界。

题　　解

1. 概念题

【题 9.1】 解释以下名词

互连网络 —— 一种由开关元件按照一定的拓扑结构和控制方式构成的网络，用来实现计算机系统中结点之间的相互连接。在拓扑上，互连网络是输入结点到输出结点之间的一

组互连或映像。

互连函数——用变量 x 表示输入,用函数 $f(x)$ 表示输出。则 $f(x)$ 表示:在互连函数 f 的作用下,输入端 x 连接到输出端 $f(x)$。它反映了网络输入端数组和输出端数组之间对应的置换关系或排列关系,所以互连函数有时也称为置换函数或排列函数。

网络规模——指互连网络中结点的个数。它表示该网络所能连接的部件的数量。

结点度——指互连网络中结点所连接的边数(通道数)。

结点距离——对于互连网络中的任意两个结点,其距离是指从一个结点出发到另一个结点终止所需要跨越的边数的最小值。

网络直径——指互连网络中任意两个结点之间距离的最大值。

等分带宽——把由 N 个结点构成的网络切成结点数相同($N/2$)的两半,在各种切法中,沿切口边数的最小值。

对称网络——从任意结点来看,网络的结构都是相同的。

静态互连网络——各结点之间有固定的连接通路且在运行中不能改变的网络。

动态互连网络——由交换开关构成、可按运行程序的要求动态地改变连接状态的网络。

虚拟通道——虚拟通道是两个结点间的逻辑连接,它由源结点的片缓冲区、结点间的物理通道以及接收结点的片缓冲区组成。

自适应寻径——通信的通路每次都根据通信资源或者网络的情况来选择,这样就可以避开拥挤的或有故障的结点,从而使网络的利用效率可以得到改进。

2. 选择题

【题 9.2】 答:B
【题 9.3】 答:B
【题 9.4】 答:C
【题 9.5】 答:B
【题 9.6】 答:A
【题 9.7】 答:A
【题 9.8】 答:D
【题 9.9】 答:D
【题 9.10】 答:B
【题 9.11】 答:B
【题 9.12】 答:B
【题 9.13】 答:D
【题 9.14】 答:C

3. 填空题

【题 9.15】 答:互连结构、开关元件、控制方式
【题 9.16】 答:可扩展性
【题 9.17】 答:互连函数、输入、输出
【题 9.18】 答:1110、0101、0110、1010

【题 9.19】　答：2、15

【题 9.20】　答：$x_{n-2}\cdots\bar{x}_i\cdots x_1 x_0 x_{n-1}$

【题 9.21】　答：$\lceil \log_2 N/2 \rceil$

【题 9.22】　答：全“0”、全“1”

【题 9.23】　答：2

【题 9.24】　答：边数(通道数)、入度、出度

【题 9.25】　答：2、1、$N-1$、1

【题 9.26】　答：开关模块、控制方式、级间互连模式

【题 9.27】　答：直送、交叉、上播、下播

【题 9.28】　答：$\log_2 N$、$N/2$、$\log_2 N \times N/2$

【题 9.29】　答：交换、移数

【题 9.30】　答：部分级

【题 9.31】　答：立方体、部分级

【题 9.32】　答：7

【题 9.33】　答：包、固定的、可变的、片

【题 9.34】　答：线路交换、包交换、存储转发、虚拟直通、虫蚀

【题 9.35】　答：虚拟直通寻径方法、阻塞流控制方法、扬弃并重发方法、阻塞后绕道传送

【题 9.36】　答：确定、自适应

【题 9.37】　答：通道数、最大传输时间

【题 9.38】　答：单播、选播、广播、会议

4. 问答题

【题 9.39】　答：①通信时延 -- 从源结点到目的结点传送一条消息所需的总时间。②网络时延 -- 通道时延与选路时延的和。它是由网络硬件特征决定的，与程序行为和网络传输状态无关。③端口带宽 -- 单位时间内从该端口传送到其他端口的最大信息量。④网络的聚集带宽 -- 网络从一半结点到另一半结点，单位时间内能够传送的最大信息量。⑤等分带宽 -- 等分带宽与等分宽度对应的切平面中，所有边合起来单位时间所能传送的最大信息量。

【题 9.40】　答：参数 n 是立方体的维数，k 是基数或者说是沿每个方向的结点数(多重性)。这两个数与网络中结点数 N 的关系为：$N=k^n$，($n=\log_k N$)。

【题 9.41】　答：控制方式是指对各个开关模块进行控制的方式，它可以有以下 3 种。

(1) 级控制：每一级的所有开关只用一个控制信号进行控制，这些开关只能同时处于同一种状态。

(2) 单元控制：每一个开关都有一个独立的控制信号，可各自处于不同的状态。

(3) 部分级控制：第 i 级的所有开关分别用 $i+1$ 个信号控制，$0 \leqslant i < n-1$，n 为级数。

【题 9.42】　答：不能实现含 C_1 的配对通信。即 0、1、4、5、8、9、C、D 不能与 2、3、6、7、A、B、E、F 之间通信。

因为 $C(x_3x_2x_1x_0)$ 与 $x_3x_2x_1x_0$ 之间，第 1 级开关为“直连”，则 x_1 为“0”的不能与 x_1 为

“1”的处理机号通信，即号为××0×的处理机不能与号为××1×的处理机配对通信。

【题9.43】 答：总线互连的复杂性最低，成本也是最低。其缺点是每台处理机可用的带宽较窄。

交叉开关是最昂贵的，因为其硬件复杂性以 n^2 上升，所以其成本最高。但是交叉开关的带宽和寻径性能最好。当网络的规模较小时，它是一种理想的选择。

多级互连网络的复杂度和带宽介于总线和交叉开关之间，是一种折中方案。其主要优点是采用模块化结构，可扩展性较好。不过，其时延随网络级数的增加而上升。另外，由于其硬件复杂度比总线高很多，其成本也不低。

【题9.44】 答：线路交换是一种消息寻径方式。在这种方式下，在传递一个信息之前，需要先建立一条从源结点到目的结点的物理通路，然后再传递信息。

优点：传输带宽较大，平均传输时延较小，而且使用的缓冲区小。因而适合于具有动态和突发性的大规模并行处理数据的传送。

缺点：需要频繁地建立源结点到目的结点的物理通路，时间开销会很大。

【题9.45】 答：存储转发是最简单的分组交换方式。在这种方式中，包是信息传递的基本单位。包从源结点经过一系列中间结点到达目的结点。存储转发要求所经过的每个中间结点都要设置一个包缓冲器，用于保存所传递的包。当一个包到达某个中间结点时，该结点先把这个包全部存储起来，然后在出口链路可用而且下一个结点的包缓冲器也可用的情况下，传递给下一个结点。

主要缺点：①包缓冲区大，不利于VLSI实现；②网络时延大，与结点距离成正比。

【题9.46】 答：在虫蚀方式中，把信息包“切割”成更小的单位——“片”，信息包中各片的传送是按流水方式进行的。当一个结点把头片送到下一个结点后，如果结点的输出链路空闲，该片立即传送到下一个结点。如果整条链路都空闲，该片就可以立即直达目的结点。当输出通路忙时，结点把该片存储到缓冲器中，直到输出链路空闲。头片送到下一个结点后，就可以把后面的各个片也依次送出，头片每向前传送一个结点，其他数据片就跟着相应地向前“蠕动”一步，就好像一条虫那样蠕动前进。一个结点一旦开始传送一个包中的头片后，这个结点就必须等待这个包的所有片都送出去后，才能传送其他包。不同包的片不能混合在一起传送。

优点：①每个结点的缓冲器较小，易于VLSI实现；②有较小的网络传输延迟；③通道共享性好，利用率高；④易于实现选播和广播通信模式。

缺点：当消息的一片被阻塞时，整个消息的所有片都将被阻塞在所在结点，占用了结点资源。

【题9.47】 答：

$N=16$，$n=4$。若要使第0号处理器与第15号处理器相连，需要经过4次混洗和3次交换，如图9.2所示。

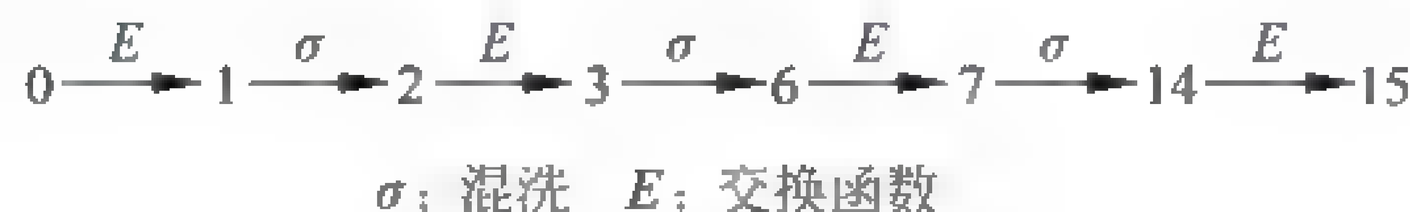


图9.2　4次混洗和3次交换

【题 9.48】 答：

(1) 线路交换寻径，传输时延公式为：

$$T = (L_t/B) \times D + L/B$$

式中，L_t 为建立路径所需的小信息包长，L 为信息包长，D 为经过的结点数，B 为带宽(以下同)。

(2) 存储转发寻径，传输时延公式为：

$$T = (L/B) \times D + L/B = (D+1) \times L/B$$

(3) 虚拟直通，传输时延公式为：

$$T = (L_h/B) \times D + L/B = (L_h \times D + L) \times L/B \approx L/B$$

式中，L_h 是消息的寻径头部的长度，一般 $L \geqslant L_h \times D$。

(4) 虫孔方式，传输时延公式为：

$$T = T_f \times D + L/B = (L_f/B) \times D + L/B = (L_f \times D + L)/B \approx L/B$$

式中，L_f 是片的长度，T_f 是片经过一个结点所需的时间，一般地，$L \gg L_f \times D$。

5. 应用题

【题 9.49】

解：(1) 共有 32 个处理机，表示处理机号的二进制地址应为 5 位。

$$E_2(12) = E_2(01100) = 01000(8)$$

$$\sigma(8) = \sigma(01000) = 10000(16)$$

$$\beta(9) = \beta(01001) = 11000(24)$$

$$\text{PM2I}_{+3}(28) = 28 + 2^3 \text{mod} 32 = 4$$

$$E_0(\sigma(4)) = E_0(\sigma(00100)) = 01001(9)$$

$$\sigma(E_0(18)) = \sigma(E_0(10010)) = \sigma(10011) = 00111(7)$$

(2) 2^n 个结点的均匀洗牌交换网的网络直径为 $2n-1$，32 个结点的均匀洗牌交换网的网络直径为 9。

从 5 号处理机发送数据到 7 号处理机，最短路径要经过 6 步：

00101→00100→01000→01001→10010→10011→00111

(3) 网络直径是 3，结点度是 9，与 2 号处理机距离最远的是 13、15、21、23 号处理机。

【题 9.50】

解：(1) 要实现 4 组 4 元交换的连接要求，要求网络输入端对输出端的连接如表 9.2 所示。

表 9.2　网络输入端对输出端的连接

十进制表示	二进制表示	十进制表示	二进制表示
0→3	0000→0011	8→11	1000→1011
1→2	0001→0010	9→10	1001→1010
2→1	0010→0001	10→9	1010→1001
3→0	0011→0000	11→8	1011→1000
4→7	0100→0111	12→15	1100→1111
5→6	0101→0110	13→14	1101→1110
6→5	0110→0101	14→13	1110→1101
7→4	0111→0100	15→12	1111→1100

由表中的二进制表示的连接要求可见,互连网络应实现的互连函数为:

$$E(x_3x_2x_1x_0)=x_3x_2\bar{x}_1\bar{x}_0=C_1(C_0(x_3x_2x_1x_0))$$

4级立方体网络在级控方式下,实现互连函数 $C_1(C_0)$ 的级控信号应为:

$$F=f_3f_2f_1f_0=0011$$

(2) 为实现4组4元交换+1组16元交换,由

输入端	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4组4元交换	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
1组16元交换	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3

可知,要求网络输入端对输出端的连接如表9.3所示。

表9.3 网络输入端对输出端的连接

十进制表示	二进制表示	十进制表示	二进制表示
0→12	0000→1100	8→4	1000→0100
1→13	0001→1101	9→5	1001→0101
2→14	0010→1110	10→6	1010→0110
3→15	0011→1111	11→7	1011→0111
4→8	0100→1000	12→0	1100→0000
5→9	0101→1001	13→1	1101→0001
6→10	0110→1010	14→2	1110→0010
7→11	0111→1011	15→3	1111→0011

由表9.3中的二进制表示的连接要求可见,互连网络应实现的互连函数为:

$$E(x_3x_2x_1x_0)=\bar{x}_3\bar{x}_2x_1x_0=C_3(C_2(x_3x_2x_1x_0))$$

4级立方体网络在级控方式下,实现互连函数 $C_3(C_2)$ 的级控信号应为:

$$F=f_3f_2f_1f_0=1100$$

【题9.51】

解:16个处理器通过 $N=16$ 的互连网络互连,根据要求的配对通道,要求网络的16个输入端对16个输出端的连接用二进制编号表示为:

0→A:0000→1010　　8→2:1000→0010

1→B:0001→1011　　9→3:1001→0011

2→8:0010→1000　　A→0:1010→0000

3→9:0011→1001　　B→1:1011→0001

4→E:0100→1110　　C→6:1100→0110

5→F:0101→1111　　D→7:1101→0111

6→C:0110→1100　　E→4:1110→0100

7→D:0111→1101　　F→5:1111→0101

可见,要求互连网络实现的互连函数为:

$$f(x_3x_2x_1x_0)=x_3\bar{x}_2x_1x_0$$

可以用 $N=16$ 的 $n=\log_2 N=4$ 级的STARAN网络来实现要求的互连,并采用级控方式,且级控信号为 $F=f_3f_2f_1f_0=1010$。因为,级控方式的STARAN网络中,若第 i 级的控制信号 $f_i=1$,则实现 C_i 置换连接,故当级控信号为 F=1010 时,STARAN网络实现的连

接为：

$$C_3(C_1(x_3x_2x_1x_0)) = \bar{x}_3x_2\bar{x}_1x_0$$

即为配对通信要求实现的互连函数。

由级控信号：

$f_i=0$：使第 i 级开关为直送状态；

$f_i=1$：使第 i 级开关为交叉状态。

得知 STARAN 网络中的开关级 K_0 和 K_2 的开关都为直送状态，开关级 K_1 和 K_3 的开关都为交叉状态。

$N=16$ 的 4 级 STARAN 网络的结构如图 9.3 所示。

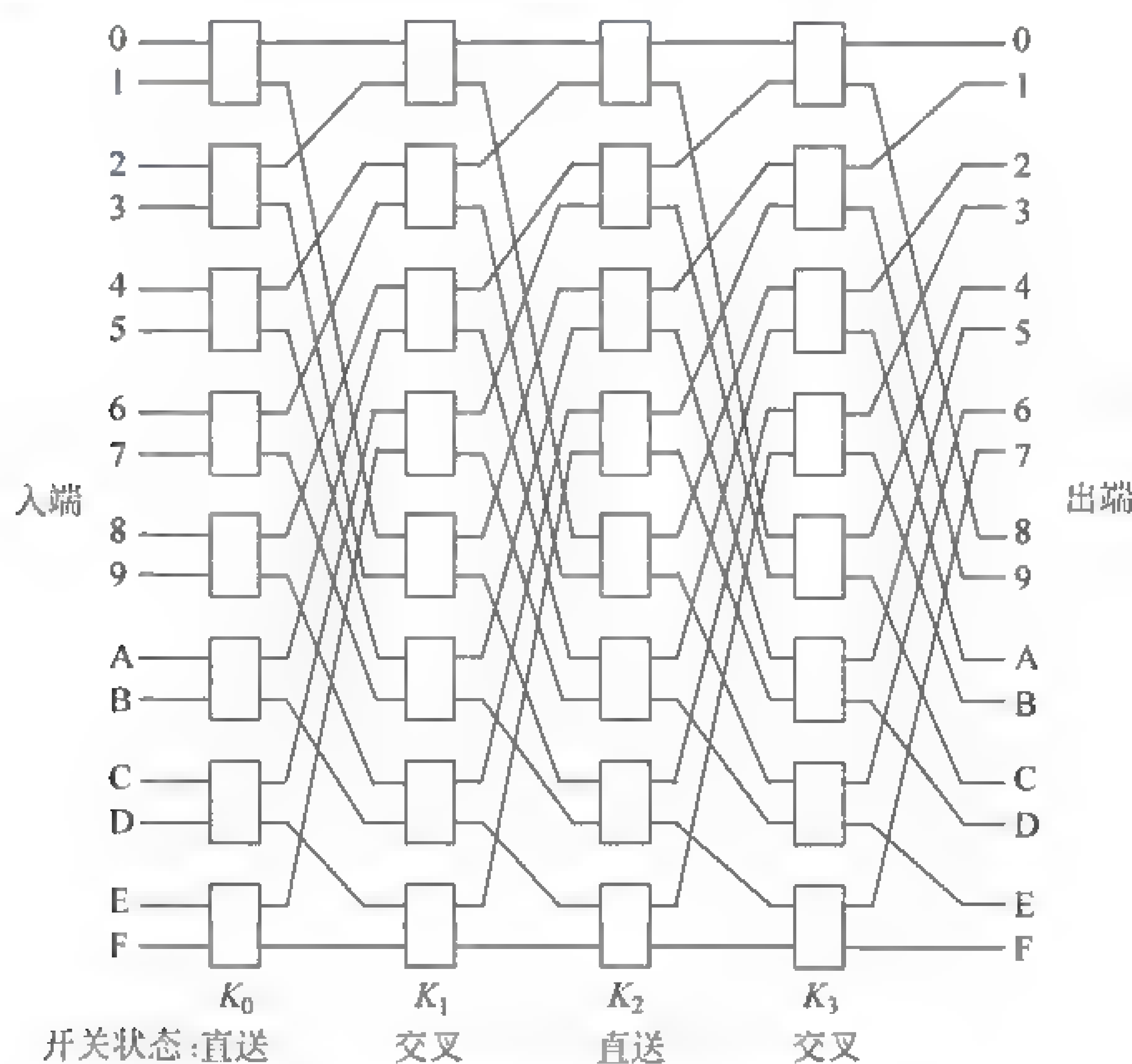


图 9.3　4 级 STARAN 网络的结构

【题 9.52】

解：Illiac 网络连接的结点数 $N-16$，组成 4×4 的阵列。每 -列的 4 个处理器互连为一个双向环，第 1～4 列的双向环可分别用循环互连函数表示为：

(0　4　8　12)　　(12　8　4　0)

(1　5　9　13)　　(13　9　5　1)

(2　6　10　14)　　(14　10　6　2)

(3　7　11　15)　　(15　11　7　3)

其中，传送方向为顺时针的 4 个单向环的循环互连函数可表示为：

$$PM2_{+2}(X) = X + 2^2 \bmod N = X + 4 \bmod 16$$

传送方向为逆时针的 4 个单向环的循环互连函数可表示为：

$$PM2_{-2}(X) = X - 2^2 \bmod N = X - 4 \bmod 16$$

16 个处理器由 Illiac 网络的水平螺线互连为一个双向环，用循环互连函数表示为：

(0　1　2　3　4　5　6　7　8　9　10　11　12　13　14　15)

(15　14　13　12　11　10　9　8　7　6　5　4　3　2　1　0)

其中,传送方向为顺时针的单向环的循环互连函数可表示为:

$$PM2_{+0}(X) = X + 2^0 \bmod N = X + 1 \bmod 16$$

传送方向为逆时针的单向环的循环互连函数可表示为:

$$PM2_{-0}(X) = X - 2^0 \bmod N = X - 1 \bmod 16$$

所以,$N=16$ 的 Illiac 网络的互连函数有4个:$PM2_{\pm 0}(X)$ 和 $PM2_{\pm 2}(X)$。

由互连函数可得任何一个处理器 i 直接与下述4个处理器双向互连:

$$i \pm 1 \bmod 16$$

$$i \pm 4 \bmod 16$$

【题 9.53】

解:已知 $N=16$ 的互连网络实现的互连函数为 $f(x_3x_2x_1x_0)=x_0x_1x_2x_3$,则由互连函数可得出16个结点之间的连接如表9.4所示。

表 9.4　16个结点之间的连接

二进制表示	十进制表示	二进制表示	十进制表示
0000→0000	0→0	1000→0001	8→1
0001→1000	1→8	1001→1001	9→9
0010→0100	2→4	1010→0101	10→5
0011→1100	3→12	1011→1101	11→13
0100→0010	4→2	1100→0011	12→3
0101→1010	5→10	1101→1011	13→11
0110→0110	6→6	1110→0111	14→7
0111→1110	7→14	1111→1111	15→15

由9.4表可见,互连网络为结点1和结点8之间实现的互连有1→8和8→1,这两个结点之间的双向互连可用循环互连函数表示为(1　8)。类似可得出互连网络的16个结点之间实现的双向互连用循环表示法表示为:

(1　8)(2　4)(3　12)(5　10)(7　14)(11　13)

【题 9.54】

解:设网络的16个输入端号序列为:[0 1 2 3 4 5 6 7 8 9 A B C D E F]

先经4组4元交换后,序列为:[3 2 1 0 7 6 5 4 B A 9 8 F E D C]

再经两组8元交换后,序列为:[4 5 6 7 0 1 2 3 C D E F 8 9 A B]

最后经一组16元交换后,得出输出端号序列为:[B A 9 8 F E D C 3 2 1 0 7 6 5 4]

由输入端号序列和输出端号序列的对应元素可知,网络实现的互连是:

(0,B)(1,A)(2,9)(3,8)(4,F)(5,E)(6,D)(7,C)

由得出的循环互连函数可以写出一般化的互连函数为:

$$C_3(C_1(x_3x_2x_1x_0)) = x_3x_2\,x_1x_0$$

其中,C 为立方体互连函数。

【题 9.55】

解:略。

【题 9.56】

解：Omega 网络使用的 2×2 开关有 4 种状态：直送、交叉、上播、下播。置换连接只使用直送和交叉状态，播送连接还需要使用上播和下播状态。分别画出实现处理机 P_6 和 P_3 的播送连接要求使用的开关状态，如果没有开关状态和开关输出端争用冲突，就可以使用播送连接。实际上，它们的播送要求没有冲突，因此，可以同时实现，同时实现的 Omega 网络开关状态图如图 9.4 所示。

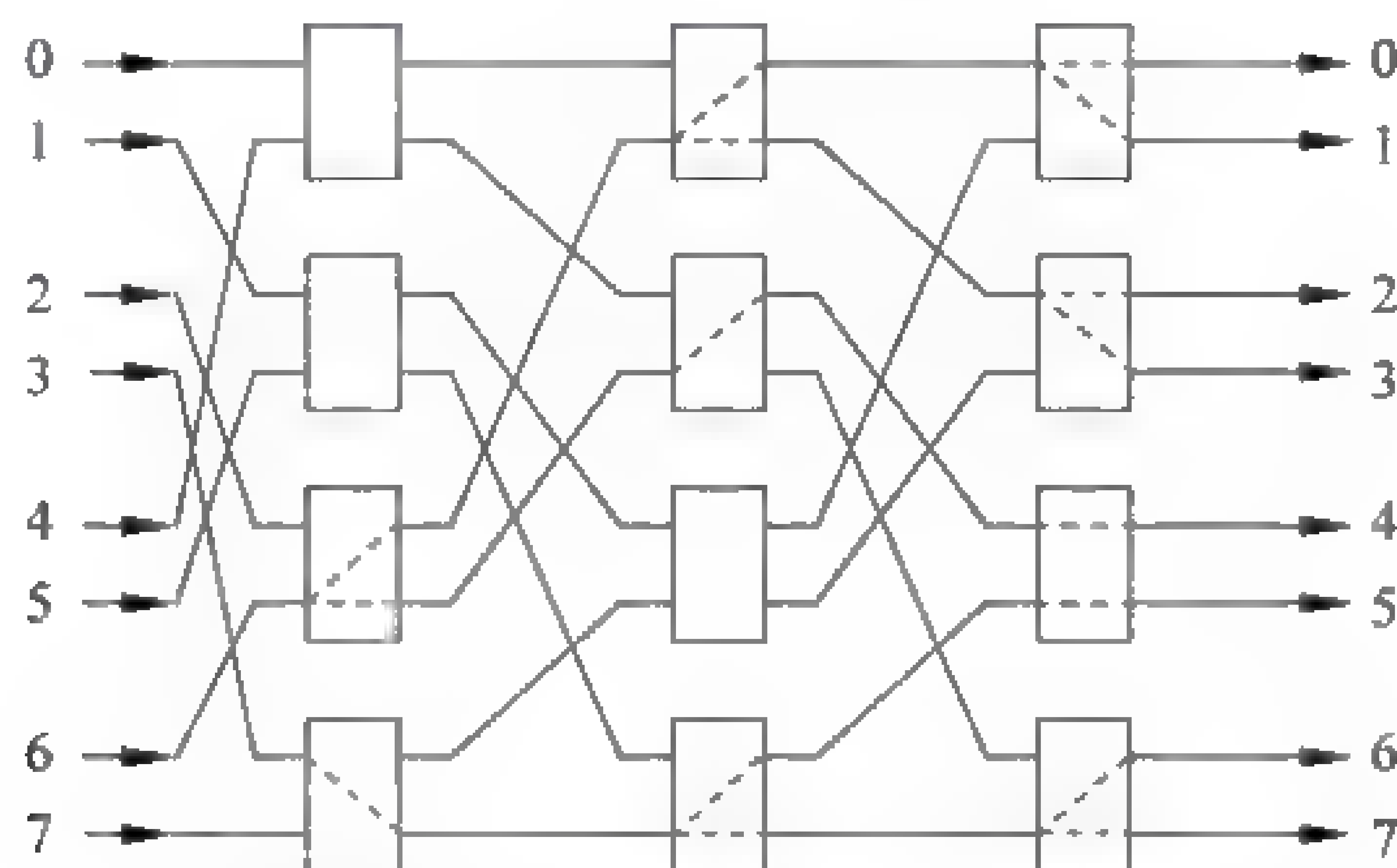


图 9.4　Omega 网络开关状态图

【题 9.57】

解：(1) 一个 $k\times k$ 开关的合法状态或合法连接有：① 一个输入端连接一个输出端，即一对一的置换连接；② 一个输入端连接多个或全部输出端，即一对多的选播连接或一对全体的广播连接。

两个或两个以上的输入端连接一个输出端是非法连接。因此，某个输出端可被连接到任意一个输入端的连接有 k 种，无论这个输出端是被置换连接还是被播送连接。

k 个输出端被连接到输入端的合法连接的数量为：

$$\underbrace{k\times k\times\cdots\times k}_{k个} = k^k$$

(2) 用 $k\times k$ 开关模块构造 N 个输入端的 Omega 网络时，开关级数为 $n=\log_k N$，每级开关模块数为 N/k，网络的开关模块总数为 $(N/k)\log_k N$。

一个 $k\times k$ 开关一对一连接的合法状态只有 k 种，所有开关都是一对一连接的合法状态才能实现一种一次使用网络的无冲突置换连接。因此，由 $(N/k)\log_k N$ 个 $k\times k$ 开关组成的 Omega 网络一次使用的无冲突置换连接函数为：

$$k^{\frac{N}{k}\log_k N} = (k^{\log_k N})^{\frac{N}{k}} = N^{\frac{N}{k}}$$

网络可以实现的置换连接数即为 N 个输出端的不同排序的排序数，即为 $N!$，所以，Omega 网使用一次实现的无冲突置换连接数占可以实现的置换连接数的比例为：

$$N^{\frac{N}{k}}/N!$$

若采用 2×2 开关模块构造的 64 个输入端的 Omega 网络，即有 $k=2$，$N=64$，则 Omega 网使用一次实现置换连接的比例为：

$$\frac{64^{32}}{64!} \approx 4.95\times 10^{-32}$$

(3) 若采用 8×8 开关模块构造 64 个输入端的 Omega 网络，即有 $k=8$，$N=64$，则

Omega 网使用一次实现置换连接的比例为：

$$\frac{64^8}{64!} \approx 3.85 \times 10^{-16}$$

(4) 若采用 8×8 开关模块构造 512 个输入端的 Omega 网络，即有 $k=8$，$N=512$，则 Omega 网使用一次实现置换连接的比例为：

$$512^{64}/512!$$

【题 9.58】

解：(1) 由 $\text{Cube}_3(x_3x_2x_1x_0)=\overline{x}_3x_2x_1x_0$，得 $\text{Cube}_3(1101)=0101$，即处理器 13 连接到处理器 5。

令 $\text{Cube}_3(x_3x_2x_1x_0)=1101$，得 $x_3x_2x_1x_0=0101$，故与处理器 13 相连的是处理器 5。

所以处理器 13 与处理器 5 双向互连。

(2) 由 $\text{PM2}_{+3}=j+2^3 \bmod 16$，得 $\text{PM2}_{+3}(13)=13+2^3=5$，即处理器 13 连接到处理器 5。

令 $\text{PM2}_{+3}(j)=j+2^3 \bmod 16=13$，得 $j=5$，故与处理器 13 相连的是处理器 5。

所以处理器 13 与处理器 5 双向互连。

(3) 由 $\text{PM2}_{-0}(j)=j-2^0 \bmod 16$，得 $\text{PM2}_{-0}(13)=13-2^0=12$，即处理器 13 连接到处理器 12。

令 $\text{PM2}_{-0}(j)=j-2^0 \bmod 16=13$，得 $j=14$，故与处理器 13 相连的是处理器 14。

所以处理器 13 连至处理器 12，而处理器 14 连至处理器 13。

(4) 由 $\sigma(x_3x_2x_1x_0)=x_2x_1x_0x_3$，得 $\sigma(1101)=1011$，即处理器 13 连接到处理器 11。

令 $\sigma(x_3x_2x_1x_0)=1101$，得 $x_3x_2x_1x_0=1110$，故与处理器 13 相连的是处理器 14。

所以处理器 13 连至处理器 11，而处理器 14 连至处理器 13。

(5) 由 $\sigma(\sigma(x_3x_2x_1x_0))=x_1x_0x_3x_2$，得 $\sigma(\sigma(1101))=0111$，即处理器 13 连接到处理器 7。

令 $\sigma(\sigma(x_3x_2x_1x_0))=1101$，得 $x_3x_2x_1x_0=0111$，故与处理器 13 相连的是处理器 7。

所以处理器 13 与处理器 7 双向互连。

【题 9.59】

解：8 个处理机实际连通的 8 个存储器的排列顺序如表 9.5 所示。

表 9.5　8 个处理机实际连通的 8 个存储器的排列顺序

输入排列(处理机)	输出排列(存储器)		
	$K_0K_1K_2=100$	$K_0K_1K_2=110$	$K_0K_1K_2=111$
0	4	6	7
1	5	7	6
2	6	4	5
3	7	5	4
4	0	2	3
5	1	3	2
6	2	0	1
7	3	1	0

【题 9.60】

解：3 种网络的性能如表 9.6 所示。

表 9.6　三种网络的性能

网络	d	D	l	$(d\times D\times l)^{-1}$	名次
3 维网络	6	6	192	1/6912	2
6 维超立方体	6	6	192	1/6912	2
带环立方体	3	9	96	1/2592	1

【题 9.61】

解：

(1) 循环移数网络中，与任意结点 $N_i(x_0x_1x_2x_3x_4x_5x_6x_7)$ 相连的结点是与其距离为 2 整数幂的结点。由此可得：

第一步：$N_1,N_2,N_4,N_8,N_{16},N_{32}$

第二步：$N_3,N_5,N_6,N_7,N_9,N_{10},N_{12},N_{14},N_{15},N_{17},N_{18},N_{20},N_{24},$
$N_{28},N_{30},N_{31},N_{33},N_{34},N_{36},N_{40},N_{48}$

第三步：$N_{11},N_{15},N_{19},N_{21},N_{22},N_{23},N_{25},N_{26},N_{27},N_{29},N_{35},N_{37},N_{38},$
$N_{39},N_{41},N_{42},N_{44},N_{46},N_{47},N_{49},N_{50},N_{52},N_{56},N_{60},N_{62},N_{63}$

(2) 所需的最少寻径步的最小上界即为网络的直径，故为：

$$\log_2 64/2=3$$

第10章　多处理机

10.1　基本要求与难点

10.1.1　基本要求

(1) 掌握有关多处理机的基本概念。
(2) 掌握并行计算机系统结构的分类,掌握多处理机的通信机制及其特点。
(3) 理解多处理机的 Cache 一致性问题,了解有哪些解决方法。
(4) 熟练掌握监听协议的基本原理和实现方法。
(5) 熟练掌握目录协议的基本原理和实现方法。
(6) 了解目录的 3 种结构。
(7) 理解处理机间的同步问题,掌握同步的实现方法,了解同步的性能问题。
(8) 理解同时多线程的概念,掌握其实现方法。了解同时多线程的性能。
(9) 了解高性能并行计算机系统结构通常可以分成哪几类,它们各有何特点。
(10) 了解大规模并行处理机 MPP。
(11) 了解多处理机 T1 和 Origin2000 的组成结构、互连方式以及特点等。

10.1.2　难点

(1) 多处理机的 Cache 一致性问题。
(2) 监听协议的基本原理和实现方法。
(3) 目录协议的基本原理和实现方法。
(4) 处理机间的同步。
(5) 同时多线程。

10.2　知识要点

10.2.1　引言

1. 并行计算机系统结构的分类

在 20 世纪 90 年代,芯片容量(即晶体管数)的增加使得人们能够在一块芯片上实现多

个处理器。这种方法最初叫片内多处理器或者单片多处理器，而现在则叫**多核**(multicore)，因为它是在一块芯片上实现多个处理器核。

可把现有的 MIMD 计算机分为两类，每一类代表了一种存储器的结构和互连策略。由于多处理机的规模大小这个概念的含义是随时间而变化的，所以我们用存储器的组织结构来区分这些机器。

第一类机器称为**集中式共享存储器结构**。这类多处理机在目前最多是由几十个处理器构成。由于处理器个数较少，各处理器可共享一个集中式的物理存储器，并通过总线或交换开关互连。因为只有单一的主存，而且这个主存相对于各处理器的关系是对称的，所以这类机器经常称为**对称式共享存储器多处理机 SMP**。这种系统结构也称为 **UMA 结构**(Uniform Memory Access)，这是因为从各处理器访问存储器所花的时间相同。SMP 结构是目前最流行的结构。

第二类是**分布式存储器多处理机**。在这类机器中，存储器在物理上是分布的。它支持构建规模较大的多处理机系统。这种结构要求有高带宽的互连网络。

将存储器分布到各结点有两个优点：①如果大多数的访存都是针对本结点的本地存储器进行的，则可以降低对存储器和互连网络的带宽要求；②对本地存储器的访问延迟时间小。分布式存储器系统结构最主要的缺点是处理器之间的通信较为复杂，且各处理器之间访问延迟较大。

2. 存储器系统结构和通信机制

1) 两种存储器系统结构和通信机制

如上所述，在大规模的多处理机中，存储器在物理上是分布于各个处理结点中的。但在逻辑地址空间的组织方式以及处理器之间通信的实现方法上，有以下两种方案。

第一种方案把物理上分离的所有存储器作为一个统一的共享逻辑空间进行编址，这样任何一个处理器都可以访问该共享空间中的任何一个单元(如果它具有访问权)，不同处理器上的同一个物理地址指向的是同一个存储单元。这类计算机被称为**分布式共享存储器系统**(**DSM**)。与 UMA 相反，DSM 计算机被称为 **NUMA**(Non Uniform Memory Access)计算机，这是因为其访存时间取决于数据在存储器中的存放位置。

另一种方案是把每个结点中的存储器编址为一个独立的地址空间，不同结点中的地址空间之间是相互独立的。每个结点中的存储器只能由本地的处理器进行访问，远程的处理器不能直接对其进行访问。

对于上述两种地址空间的组织方案，分别有相应的**通信机制**。对于共享地址空间的计算机系统来说，是采用**共享存储器通信机制**。处理器之间是通过用 load 和 store 指令对相同存储器地址进行读/写操作来实现的。而对于采用多个独立地址空间的计算机系统来说，数据通信要通过在处理器之间显式地传递消息来完成，这称为**消息传递通信机制**。

在消息传递多处理机中，处理器之间是通过发送消息来进行通信的，这些消息请求进行某些操作或者传送数据。不同多处理机所提供的消息传递机制可能差别很大，为了便于程序移植，人们提出了标准的消息传递库(例如，MPI)，这为编程人员实现消息传递提供了有力的支持，使他们能很容易地进行消息通信。

2) 不同通信机制的优点

每种通信机制各有自己的优点,共享存储器通信的主要优点如下。

(1) 与常用的对称式多处理机使用的通信机制兼容。

(2) 当处理器之间通信方式复杂或在执行过程中动态变化时,采用共享存储器通信,编程容易,同时在简化编译器设计方面也占有优势。

(3) 采用大家所熟悉的共享存储器模型开发应用程序,而把重点放到解决对性能影响较大的数据访问上。

(4) 当通信数据量较小时,通信开销较低,带宽利用较好。

(5) 可以通过采用 Cache 技术来减少远程通信的频度。这是通过对所有数据(包括共享的和私有的)进行 Cache 缓冲来实现的。在后面我们将看到,Cache 不仅能减少访问共享数据的延迟,而且能减少对共享数据的访问冲突。

消息传递通信机制的主要优点如下。

(1) 硬件更简单。特别是在与可扩放共享存储器实现方案相比时更是如此。

(2) 通信是显式的,因此更容易搞清楚何时发生通信以及通信开销是多少。

(3) 显式通信可以让编程者重点注意并行计算的主要通信开销,使之有可能开发出结构更好、性能更高的并行程序。

(4) 同步很自然地与发送消息相关联,能减少不当的同步带来错误的可能性。

最初的分布式存储器计算机均采用消息传递机制,因为它显然比较简单。但近些年来,特别是20世纪90年代后半期以来设计的计算机几乎都采用共享存储器通信。

尽管现在通过总线连接的对称式存储器计算机在市场上仍占主导地位,但从长远来看,在技术上的趋势是朝着中等规模的分布式共享存储器计算机方向发展。

3. 并行处理面临的挑战

并行处理面临着两个重要的挑战,一个是程序中的并行性有限,另一个是相对较大的通信开销。有限的并行性使计算机要达到很高的加速比十分困难。这可以用 Amdahl 定律进行解释。第二个挑战主要是指多处理机中远程访问的较大延迟。在现有的计算机中,处理器之间的数据通信大约需要50~1000个时钟周期,这主要取决于通信机制、互连网络的种类和机器的规模。

应用程序中并行性不足的问题主要是通过采用并行性更好的算法来解决。而减少远程访问延迟则既可以依靠系统结构来实现,也可以通过编程技术来实现。我们既可以采用硬件的方法,例如用 Cache 来缓冲共享数据;也可以采用软件的方法对数据重新进行组织,使得更多的访问变成局部访问。我们还可以采用预取或多线程技术来减少延迟的影响。

反映并行程序性能的一个重要的度量是**计算/通信比率**。如果比值较高,就意味着应用程序中相对于每次数据通信要进行较多的计算。如前所述,通信在并行计算中的开销是很大的,因而较高的计算/通信比率十分有益。通常状况下,计算/通信比率随着处理的数据规模的增大而增加,随着处理器数目的增加而减少。这告诉我们用更多的处理器来求解一个固定大小的问题会导致不利因素的增加,因为处理器之间通信量加大了。这同时也告诉我们增加处理器时应该调整数据的规模,从而使通信的时间尽量保持不变。

10.2.2　对称式共享存储器系统结构

自 20 世纪 80 年代以来，随着微处理器逐渐成为主流，人们设计出了许多通过总线共享一个单独物理存储器的小规模多处理机。由于大容量 Cache 很大程度地降低了对总线带宽的要求，当处理机规模较小时，这种计算机十分经济。以往的这种计算机一般是将 CPU 和 Cache 做在一块板上，然后插入底板总线。后来，每块板上的处理器数目达到了 4 个，而近些年，则能在一个单独的芯片上实现 2～8 个处理器核。例如，Sun 公司在 2006 年发布的 T1 就是一个 8 核的多处理器。

对称式共享存储器系统结构一般都支持对共享数据和私有数据的 Cache 缓存。私有数据是指只供一个处理器使用的数据，而共享数据则是指供多个处理器共同使用的数据。处理器之间可以通过读/写共享数据来实现通信。

私有数据进入 Cache，使得处理器对它们的访问可以在 Cache 中完成，从而减少平均访存时间和减少对存储器带宽的要求。当允许共享数据进入 Cache 时，共享数据可能会在多个 Cache 中被复制，这样相应的处理器就可以在自己的 Cache 中找到这些数据。这样做不仅可以减少访存时间和对存储器带宽的要求，而且还可以减少多个处理器同时读取共享数据所产生的冲突。不过，共享数据进入 Cache 也带来了一个新的问题，即 Cache 的一致性问题。

1. 多处理机 Cache 一致性

如果允许共享数据进入 Cache，那么就可能出现多个处理器的 Cache 中都有同一存储器块的副本，当其中某个处理器对其 Cache 中的数据进行修改后，就会使得其 Cache 中的数据与其他 Cache 中的数据不一致。这就是**多处理机的 Cache 一致性**问题。

对于一致性，我们可以有这样的说法：如果对某个数据项的任何读操作均可得到其最新写入的值，则认为这个存储系统是一致的。

如果一个存储器满足以下三点，则称该存储器是一致的。

(1) 处理器 P 在对存储单元 X 进行一次写之后又对 X 进行读，在这读和写之间没有其他处理器对 X 进行写，则 P 读到的值总是刚写进去的值。

(2) 处理器 P 对存储单元 X 进行写之后，另一处理器 Q 对 X 进行读，在这读和写之间没有其他对 X 的写，则 Q 读到的值应为 P 写进去的值。

(3) 对同一存储单元的写是串行化的。即任意两个处理器对同一存储单元的两次写，从各个处理器的角度看来顺序都是相同的。

第一条属性保证了程序顺序，即使在单处理机中也要求如此。第二条属性给出了存储器一致性的概念。如果一个处理器不断地读取到旧的数据，我们可以肯定地说这个存储器是不一致的。

把写操作串行化，能使得对同一存储器单元所进行的写操作顺序在所有处理器看来都是相同的。这种属性称为**写串行化**。

为了简化起见，在后面的讨论中做以下假设：①直到所有的处理器均看到了写的结果，这个写操作才算完成；②处理器的任何访存均不能改变写的顺序。就是说，允许处理器对

读进行重排序,但必须以程序规定的顺序进行写。

2. 实现一致性的基本方案

在支持Cache一致性的多处理机中,Cache实现了共享数据的迁移和复制功能。**共享数据的迁移**是把远程的共享数据复制一份,迁入本地Cache供本处理器使用,从而减少了对远程共享数据的访问延迟,也减少了对共享存储器带宽的要求。**共享数据的复制**则是把多个处理器需要同时读取的共享数据在这些处理器的本地Cache中各存放一个副本。共享数据的迁移和复制对于提高访问共享数据的性能来说是非常重要的。

在多个处理器中用来维护一致性的协议称为**Cache一致性协议**。实现Cache一致性协议的关键是跟踪共享数据块的状态。目前有两类协议,它们采用了不同技术来跟踪共享数据的状态。

(1) 目录式协议——物理存储器中数据块的共享状态被保存在一个称为目录的地方。目录式协议的实现开销比监听式协议的稍微大一些,但可用于实现更大规模的多处理机。

(2) 监听式协议——当物理存储器中的数据块被调入Cache时,其共享状态信息与该数据块一起放在该Cache中。当某个Cache需要访问存储器时,它会把请求放到总线上广播出去,其他各个Cache控制器通过监听总线来判断它们是否有总线上请求的数据块。如果有,就进行相应的操作。

可以采用两种方法来解决上述的Cache一致性问题:写作废协议,写更新协议。

(1) 写作废协议:在处理器对某个数据项进行写入之前,它必须先拥有对该数据项的唯一的访问权。这是通过把所有其他Cache中的副本全部作废来实现的。它是目前最常用的协议,无论是采用监听协议还是采用目录协议都是如此。

如果两个处理器要同时进行写操作,则它们竞争访问权。它们中只有一个会在竞争中获胜——获得访问权,而另一个处理器中的副本以及其他处理器中的副本(如果有的话)都会被作废。竞争失败的处理器要完成写操作,就必须先获得一份新的数据副本。该副本已经包含更新后的数据。显然,这种协议保证了写操作的串行化。

(2) 写更新协议。在这种协议中,当一个处理器对某数据项进行写入时,它把该新数据广播给所有其他Cache。这些Cache用该新数据对其中的副本(如果有的话)进行更新。当然,如果知道其他Cache中都没有相应的副本,就不必进行广播和更新。这样处理能够减少实现该协议所需的带宽。

写更新和写作废协议在性能上的主要差别如下。

(1) 在对同一个数据进行多次写操作而中间无读操作的情况下,写更新协议需进行多次写广播操作,而写作废协议只需一次作废操作。

(2) 在对同一Cache块的多个字进行写操作的情况下,写更新协议对于每一个写操作都要进行一次广播,而写作废协议仅在第一次写时进行作废操作即可。写作废是针对Cache块进行操作,而写更新则是针对字(或字节)进行。

(3) 考虑从一个处理器A进行写操作后到另一个处理器B能读到该写入数据之间的延迟时间。在写更新协议中,这个延迟时间比较小,因为它在进行写操作时,立即更新了所有其他Cache中的副本,包括Cache B中的副本(假设有此副本)。而在写作废协议中,由于

在处理器A进行写操作时已经作废了Cache B中的副本，所以当处理器B进行读操作时需要等待，直到新的副本被调入Cache并送给它。

后面我们只关注写作废协议。

3. 监听协议的实现

实现监听协议的关键有3个方面：①处理器之间通过一个可以实现广播的互连机制相连。通常采用的是总线。②当一个处理器的Cache响应本地CPU的访问时，如果它涉及全局操作（例如作废等），其Cache控制器就要在获得总线的控制权后，在总线上发出相应的消息。③所有处理器都一直在监听总线，当它们检测到总线上的地址在它们的Cache中有副本时，则响应该消息，并进行相应的操作。

获取总线控制权的顺序性保证了写操作的串行化，因为当两个处理器要同时对同一数据块进行写操作时，必然是只有其中一个处理器先获得总线控制权，并作废所有其他处理器上的相关副本。另一处理器要等待前一个处理器的写操作完成后，再排队竞争总线控制权。这保证了写操作严格地按顺序进行。所有的一致性协议都要采用某种方法来保证对同一个Cache块的写访问的串行化。

虽然不同的监听协议在具体实现上有些差别，但在许多方面是相同的。Cache发送到总线上的消息主要有以下两种。

（1）RdMiss——读不命中。

（2）WtMiss——写不命中。

RdMiss和WtMiss分别表示本地CPU对Cache进行读访问和写访问时不命中，这时都需要通过总线找到相应数据块的最新副本，然后调入本地Cache。对于写直达Cache来说，由于所有写入的数据都同时被写回存储器，所以其最新值总可以从存储器中找到。而对于写回法Cache来说，难度就大一些了，因为这个最新副本有可能是在其他某个处理器的Cache中（尚未写回存储器）。在这种情况下，将由该Cache向请求方处理器提供该块，并终止由RdMiss或WtMiss所引发的对存储器的访问。

有的监听协议还增设了一条Invalidate消息，用来通知其他各处理器作废其Cache中相应的副本。Invalidate和WtMiss的区别在于Invalidate不引起调块。

在采用写回法的多Cache中，我们可以直接利用“修改位”来实现一致性。将一个数据进行写入时，只写入Cache，而不直接写回存储器。这时这个块的修改位被置位，表示该块中保存的是整个系统中唯一的最新副本，存储器中的副本是过时了的，而且所有其他Cache中也没有其副本。

每个处理器（实际上是Cache控制器）都监听其他处理器放到总线上的地址，如果某个处理器发现它拥有被请求数据块的一个最新副本，它就把这个数据块送给发出请求的处理器。与写直达法相比，尽管写回法在实现的复杂度上有所增加，但由于写回法Cache所需的存储器带宽较低，它在多处理机实现上仍很受欢迎。在后面的讨论中，我们只考虑写回法Cache。

Cache本来就有的标识（tag）可直接用来实现监听。通过把总线上的地址和Cache内的标识进行比较，就能找到相应的Cache块（如果有），然后对其进行相应的处理。每个块的有效位使得我们能很容易地实现作废机制。当要作废一个块时，只需将其有效位置为无效

即可。对于CPU读不命中的情况,处理比较简单,Cache控制器向总线发RdMiss消息,并启动从主存的读块操作,准备调入Cache。当然,如果存储器中的块不是最新的,最新的副本是在某个Cache中,就要由该Cache提供数据,并终止对存储器的访问。

对于写操作来说,我们希望能够知道其他处理器中是否有该写入数据的副本,因为如果没有,那么就不用把这个写操作放到总线上,从而减少所需要的带宽以及这个写操作所花的时间。这可以通过给每个Cache块增设一个共享位来实现。该共享位用来表示该块是被多个处理器所共享(共享位为"1"),还是仅被某个处理器所独占(共享位为"0")。拥有该数据块的唯一副本的处理器通常被称为该块的**拥有者**。

当一个块处于独占状态时,其他处理器中没有该块的副本,因此不必向总线发Invalidate消息。否则就是处于共享状态,这时要向总线发Invalidate消息,作废所有其他Cache中的副本,同时将本地Cache中该块的共享标志位置"0"。如果后面又有另一处理器再读这个块,则其状态将再次转化为共享。由于每个Cache都在监听总线上的消息,所以它们知道什么时候另一个处理器请求访问该块,从而把其状态改为共享。

每一次总线操作都要检查Cache中的地址标识,这会打扰处理器对Cache的访问。必须设法减少这种打扰。一种方法是设置两套标识,分别用于处理来自CPU的访问和来自总线的访问。当然,Cache不命中时,处理器要对两套标识进行操作。类似地,如果监听到了一个相匹配的地址,也要对两套Cache的标识进行操作。

在多级Cache中,还可以采用另一种方法,即把监听的操作请求交给第二级Cache来处理。由于处理器只在第一级Cache不命中时才会访问第二级Cache,而第一级Cache的命中率往往都很高,所以这种方法是可行的。不过,当监听机制在第二级Cache中发现相匹配的项目时,就会与处理器争用第一级Cache。在监听机制获得使用权后,要修改状态并可能需要访问第一级Cache中的数据。此外,这两级Cache必须满足包容关系,即第一级Cache中的内容是第二级Cache中内容的一个子集。

10.2.3 分布式共享存储器系统结构

1. 目录协议的基本思想

监听协议直接利用了系统中已经存在的总线和Cache中的状态位。因而它具有实现容易、成本较低的优点。然而,当系统的规模变大时,它又是个致命的弱点。大量的总线广播操作会使得总线很快就成为系统的瓶颈。广播和监听的机制使得监听一致性协议的可扩放性很差。

为了实现较大规模的可扩放的共享存储器多处理机系统,需要寻找新的一致性协议来代替监听协议,这就是前面提过的目录协议。另外,总线的可扩放性不好,可以改用可扩放性更好的互连网络。互连网络能很高效地实现点到点的通信。目录协议采用了一个集中的数据结构——目录。对于存储器中的每一个可以调入Cache的数据块,在目录中设置一条目录项,用于记录该块的状态以及哪些Cache中有副本等相关信息。这样,对于任何一个数据块,都可以快速地在唯一的一个位置中找到相关的信息。这使得目录协议避免了广播操作。

目录法常采用位向量的方法来记录哪些 Cache 中有副本，该位向量中的每一位对应于一个处理器。这个位向量的长度与处理器的个数成正比。为便于讨论，后面将把由位向量指定的处理机的集合称为共享集 S。目录协议根据该项目中的信息以及当前要进行的访问操作，依次对相应的 Cache 发送控制消息，并完成对目录项信息的修改。此外，还要向请求处理器发送响应信息。

为了提高可扩放性，可以把存储器及相应的目录信息分布到各结点中。每个结点的目录中的信息是对应于该结点存储器中的数据块的。这使得对于不同目录项的访问可以在不同的结点中并行进行。当处理器进行访存操作时，如果其地址落在本地存储器的地址范围中，就是本地的，否则就是远程的，这是由结点内的控制器根据访问地址来判定的。

对于目录法来说，最简单的实现方案是对于存储器中每一块都在目录中设置一项。在这种情况下，目录中的信息量与 $M\times N$ 成正比。其中，M 表示存储器中存储块的总数量，N 表示处理器的个数。由于 $M=K\times N$，其中，K 是每个处理机中存储块的数量，所以如果 K 保持不变，则目录中的信息量就与 N^2 成正比。显然这种方法的可扩放性不好。只有在处理器个数比较少的情况下才是可行的。

当处理器数量较多时，需要采用可扩放性更好的方法，例如只给那些已经进入 Cache 的块（而不是所有的块）设置目录项，或者让每个目录项的位数固定。后面将进一步介绍这些方法。

与监听协议类似，目录协议也必须实现两种基本操作：处理读不命中和处理对共享、干净块的写。对一个共享块的写不命中的处理可用这两个操作组合而成。

在目录协议中，存储块的状态有以下三种。

（1）未缓冲（Uncached）——该块尚未被调入 Cache。所有处理机的 Cache 中都没有这个块的副本。

（2）共享（Shared）——该块在一个或多个处理机上有这个块的副本，且这些副本与存储器中的该块相同。

（3）独占（Exclusive）——仅有一个处理机有这个块的副本，且该处理机已经对其进行了写操作，所以其内容是最新的，而存储器中该块的数据已过时。这个处理机称为该块的拥有者。

为了提高实现效率，我们在每个 Cache 中还跟踪记录每个 Cache 块的状态。

在目录法中，每个 Cache 中的 Cache 块的状态及其转换与前面监听法的情况相同。只是在状态转换时所进行的操作有些不同。

本地结点是指发出访问请求的结点。该结点中的处理机 P 发出了一个地址为 K 的访存请求。宿主结点是指包含所访问的存储单元及其目录项的结点，它包含地址 K 的存储单元及相应的目录项。而拥有相应存储块副本的结点则称为**远程结点**。因为物理地址空间是静态分布的，所以对于某一给定的物理地址，包含其存储单元及目录项的结点是确定且唯一的。该地址的高位指出结点号，而低位则表示在相应结点的存储器内的偏移量。

本地结点和宿主结点可以是同一个结点，这时所访问的地址单元就在本地结点的存储器中。远程结点可以和宿主结点是同一个结点，也可以和本地结点是同一个结点。在这些情况下，基本协议不需要变动，只是结点之间的消息变成了结点内的消息。

2. 目录协议实例

基于目录的协议中Cache的基本状态与监听协议中的相同,Cache块状态转换的操作在实质上也与监听模式相同。在监听协议中,写失效操作要在总线上进行广播,现在则由从宿主结点的取数据和目录控制器有选择地发出写作废操作来代替。与监听协议相同,当对Cache块进行写操作时,该Cache块必须是处于独占状态。另外,对于任何一个处于共享状态的块来说,其宿主存储器中的内容都是最新的。

在基于目录的协议中,目录承担了一致性协议操作的主要功能。目录中的状态有:共享、未缓冲和专有,它们用于指出相应的存储块所处的状态。未缓冲是指该存储块在所有的处理器结点中都没有副本;共享是指该块在多个处理器结点中有副本,而且只是可读的;专有是指该块只在一个结点中有副本,并且在该结点中是可写的。除了每个块的状态外,目录项还用位向量记录拥有其副本的处理器集合。这个集合称为共享集合。对目录表的请求处理需更新共享集合。发往一个目录的消息会产生两种不同类型的动作:更新目录状态和发送消息以完成所请求的操作。

目录可能接收到三种不同的请求:读不命中、写不命中或数据写回。可以假设这些操作是原子的。为了进一步理解对目录的操作,下面分析一下各个状态下所接收到的请求和相应的操作。

(1) 当一个块处于未缓冲状态时,对该块发出的请求及处理操作如下。

① 读不命中——将存储器数据送往请求方处理器,且该处理器成为该块的唯一共享结点,本块的状态变成共享。

② 写不命中——将存储器数据送往请求方处理器,该块的状态变成独占,表示该块仅存在唯一的有效副本。其共享集合仅包含该处理器,指出该处理器是其拥有者。

(2) 当一个块处于共享状态时,其在存储器中的数据是当前最新的,对该块发出的请求及处理操作如下。

① 读不命中——将存储器数据送往请求方处理器,并将其加入共享集合。

② 写不命中——将数据送往请求方处理器,对共享集合中所有的处理器发送写作废消息,且将共享集合改为仅含有该处理器,该块的状态变为独占。

(3) 当某块处于独占状态时,该块的最新值保存在共享集合所指出的唯一处理器(拥有者)中。有以下三种可能的目录请求。

① 读不命中——将"取数据"的消息发往拥有者处理器,使该块的状态转变为共享,并将数据送回宿主结点写入存储器,进而把该数据送回请求方处理器,将请求方处理器加入共享集合。此时共享集合中仍保留原拥有者处理器(因为它仍有一个可读的副本)。

② 写不命中——该块将有一个新的拥有者。给旧的拥有者处理器发送消息,要求它将数据块送回宿主结点写入存储器,然后再从该结点送给请求方处理器。把请求处理器加入共享者集合,使之成为新的拥有者。该块的状态仍旧是独占。

③ 数据写回——当一个块的拥有者处理器要从其Cache中把该块替换出去时,必须将该块写回其宿主结点的存储器中,从而使存储器中相应的块中存放的数据是最新的(宿主结点实际上成为拥有者),该块的状态变成未缓冲,其共享集合为空。

实际计算机中采用的目录协议要做一些优化。比如对某个独占块发出读或写不命中

时，该块将先被送往宿主结点存入存储器，然后再被送往请求结点，而实际中的计算机很多都是将数据从拥有者结点直接送该请求结点(同时写回宿主结点中的存储器)。

基于目录的Cache一致性协议是完全由硬件实现的。当然，也可以用软硬结合的办法实现，即将一个可编程协议处理机嵌入到一致性控制器中，这样既减少了成本，又缩短了开发周期。这是因为可编程协议处理机可以根据实际应用需要很快开发出来，而一致性协议处理中的异常情况可完全交给软件执行。这种软硬结合实现Cache一致性的代价是损失了一部分效率。

到目前为止讨论的一致性协议都做了一些简化的假设，实际中的协议必须处理以下两个实际问题：操作的非原子性和有限的缓存。操作的非原子性产生实现的复杂性，有限的缓存可能导致死锁问题。设计者面临的一个问题是通过非原子的操作和有限的缓存设计出一种正确的且无死锁的协议，这些因素是所有并行机面临的基本问题。

3. 目录的3种结构

不同目录协议的主要区别主要有两个，一个是所设置的存储器块的状态及其个数不同，另一个则是目录的结构。根据目录的结构，可以把目录协议分成三类：全映像目录、有限映像目录和链式目录。

1) 全映像目录

前面介绍的协议都是基于全映像目录的。在这种结构中，每一个目录项都包含一个N位(N为处理机的个数)的位向量，位向量的每一位对应于一个处理机。当位向量中的值为"1"时，就表示它所对应的处理机有该数据块的副本；否则就表示没有。在这种情况下，共享集合由位向量中值为"1"的位所对应的处理机构成。

全映像目录的处理比较简单，速度也比较快。但它的存储空间的开销很大。目录项的数目与处理机的个数N成正比，而目录项的大小(位数)也与N成正比，因此目录所占用的空间与N^2成正比。这种目录结构的可扩放性很差。

2) 有限映像目录

有限映像目录是对全映像目录的改进，为的是提高其可扩放性和减少目录所占用空间。其核心思想是采用位数固定的目录项目，这是通过对同一数据块在所有Cache中的副本总数进行限制来实现的。例如，限定为常数m，则目录项中用于表示共享集合所需的二进制位数为：$m\times\lceil\log_2 N\rceil$。人们发现，根据局部性原理，一般来说$m<<N$，这样就能大大地减小目录存储器的规模。这种目录所占用的空间与$N\times\lceil\log_2 N\rceil$成正比。

有限映像目录的缺点是：当同一数据的副本个数大于m时，必须做特殊处理。当目录项中的m个指针都已经全被占满，而某处理机又需要新调入某一块时，就需要在m个指针中选择一个，将之驱逐，以便腾出位置，存放指向新调入块的处理机的指针。

3) 链式目录

链式目录是用一个目录指针链表来表示共享集合。当一个数据块的副本数增加(或减少)时，其指针链表就跟着变长(或变短)。由于链表的长度不受限制，因而带来了以下优点：既不限制副本的个数，又保持了可扩展性。

链式目录有两种实现方法：单链法和双链法。链式目录比前两种结构都复杂不少，但它带来了可扩展性，这是前两种方法所无法实现的。链式目录的指针的位数与$\lceil\log_2 N\rceil$成

正比，但每个数据块的链表中的指针数目与处理机的个数无关。在目录所占用的空间方面，链式目录与有限映像目录类似，也是与 $N\times\lceil\log_2 N\rceil$ 成正比。

10.2.4　同步

在大规模计算机或进程竞争激烈的情况下，同步可能会成为性能的瓶颈，导致较大的延迟开销。同步机制通常是在硬件提供的同步指令的基础上，通过用户级软件例程来建立的。

1. 基本硬件原语

在多处理机中实现同步，所需的主要功能是一组能以原子操作的方式读出并修改存储单元的硬件原语。基本硬件原语有几种形式可供选择，它们都能以原子操作的方式读/修改存储单元，并指出所进行的操作是否以原子的方式进行。

用于构造同步操作的一个典型操作是**原子交换**，它的功能是将一个存储单元的值和一个寄存器的值进行交换。我们来看看如何用它来构造一个基本的同步操作。假设我们是要构造这样一个简单的锁：其值为0表示锁是开的(可用)，为1表示已上锁(不可用)。当处理器要给该锁上锁时，是将对应于该锁的存储单元的值与存放在某个寄存器中的1进行交换。如果别的处理器早已上了锁，则交换指令返回的值为1，否则为0。在后一种情况下，该锁的值会从0变成1，即上了锁。这样，其他竞争的交换指令的返回值就不会是0。

还有一些别的原语可用来实现同步，它们均具有这样的关键属性：能指出是否以原子的方式读出并更新存储单元值。**测试并置定**是其中之一，在许多以往的多处理机中都有这个操作。其功能是先测试一个存储单元的值，如果符合条件则修改其值。例如，可以定义一个操作来检测某个存储单元值是否为0，是则置1。其使用方法跟前面介绍的原子交换类似。另一个同步原语是**读取并加1**，它返回存储单元的值并自动增加该值。

然而，要在一条指令中完成上述全部操作会有一些困难，因为它要在一条不可中断的指令中完成一次存储器读和一次存储器写，而且在这一过程中不允许进行其他的访存操作，而且还要避免死锁。

现在一些计算机上用到的原子方式的读/修改方法略有不同，它们采用一对指令而不是一条指令来实现上述同步原语。这种方法是在第二条指令返回一个值，通过该值可以判断该指令对的执行结果是否相当于一个原子操作。所谓相当于原子操作是指所有其他处理器进行的操作或者是在该指令对之前，或者是在该指令对之后进行，不存在在这两条指令之间进行的操作。

该指令对由两条特殊的指令构成，一条是特殊的load指令，称为**LL(Load Linked)**，另一条是特殊的store指令，称为**SC(Store Conditional)**。如果由LL指明的存储单元的内容在SC对其进行写之前已被其他指令改写过，则第二条指令SC执行失败；如果在两条指令间进行切换也会导致SC执行失败。SC将返回一个值来指出该指令操作是否成功，如果执行成功返回1，否则返回0。LL则返回该存储单元初始值。下面这一段程序实现对由R1指出的存储单元进行原子交换操作。

```
try:  OR    R3, R4, R0         //R4中为交换值.把该值送入R3
      LL    R2, 0(R1)          //把单元0(R1)中的值取到R2
```

```
        SC     R3, 0(R1)           //若 0(R1)中的值与 R3 中的值相同,则置 R3 的值
                                   //为 1,否则置为 0
        BEQZ   R3, try             //存失败(R3 的值为 0)则转移
        MOV    R4, R2              //将取的值送往 R4
```

最终 R4 和由 R1 指向的单元值进行原子交换,在 LL 和 SC 之间如有别的处理器插入并修改了存储单元的值,SC 将返回 0 并存入 R3 中,从而使这段程序再次执行。

LL/SC 机制的一个优点是可用来构造别的同步原语。

2. 用一致性实现锁

有了原子操作,就可以采用多处理机的一致性机制来实现旋转锁。旋转锁是指处理器不停地请求获得使用权的锁。处理器围绕该锁反复执行循环程序,直到获得该锁。旋转锁适合于这样的场合:锁被占用的时间很少,在获得锁后加锁过程延迟很小。因为旋转锁会把处理器绑定在循环等待获得锁的使用权中,所以在有些情况下不适合使用。

在无 Cache 一致性机制的条件下,最简单的实现方法是把锁变量保存在存储器中,处理器可以不断地通过一个原子操作来请求其使用权,比如利用原子交换操作,并测试返回值从而知道锁的使用情况。释放锁的时候,处理器只需简单地将锁置为 0。下面这段程序用原子交换操作对旋转锁进行加锁,R1 中存放的是该旋转锁的地址。

```
         DADDIU  R2, R0, #1
lockit:  EXCH    R2, 0(R1)          //原子交换
         BNEZ    R2, lockit         //若 R2 的内容不为 0,则表示已经被其他程序上锁。
                                    //继续旋转等待
```

如果计算机支持 Cache 一致性,就可以将锁调入 Cache,并通过一致性机制使锁值保持一致。这样做有两个好处:第一,可使"环绕"的进程(对锁进行不停的测试和请求占用的小循环)只对本地 Cache 中的锁(副本)进行操作,而不用在每次请求占用锁时都进行一次全局的存储器访问;第二个好处是可利用访问锁时所具有的局部性,即处理器最近使用过的锁不久又会使用,这种状况下锁可驻留在那个处理器的 Cache 中,大大减少了获得锁所需要的时间。

要获得第一条好处,需对上面简单的旋转锁程序进行一些改动,上面循环中每次交换均需一次写操作,如果有多个处理器都同时请求加锁,则大多数写都会导致写不命中,因为每个处理器都想以独占的方式获得锁变量。因此应该对旋转锁程序进行改进,使得它只对本地 Cache 中锁的副本进行读取和检测,直到发现该锁已经被释放。然后,该程序立即进行交换操作,去跟在其他处理器上的进程争用该锁变量。这些进程也在以同样的方式"旋转等待"该锁。所有这些进程都采用交换指令来从锁变量读出原来的值,并把 1 写入锁变量。只有一个进程获胜 -- 获得锁的占有权,该进程看到锁变量原来的值是 0。其余的进程都是失败者,虽然它们也把锁变量置 1,但因它们所看到的锁变量本来就是 1,所以等于没做什么。它们还要继续"旋转等待"。获得占有权的处理器在执行完其代码后,将锁变量置 0 以释放它,其他"旋转等待"该锁的进程又开始争用。下面是修改后的旋转锁程序。

```
lockit:   LD       R2, 0(R1)        //取锁值
          BNEZ     R2, lockit       //如果锁没有被释放,继续"旋转等待"
```

```
        DADDIU  R2, R0, #1       //置 R2 为 1
        EXCH    R2, 0(R1)        //交换
        BNEZ    R2, lockit       //如果锁没有被释放,继续"旋转等待"
```

让我们分析一下这种旋转锁是怎样使用 Cache 一致性机制的。一旦一个拥有旋转锁的处理器使用完毕,并写入 0 来释放该锁,所有其他 Cache 中的对应块均被作废,必须取新的值来更新它们所拥有的锁的副本。其中一个处理器的 Cache 会先获得被释放了的锁的值(0),并进行交换操作。当别的 Cache 不命中处理完后,它们会发现该锁已经被加了锁,所以又必须不停地环绕测试。

这个例子也说明了 LL/SC 原语的另一个优点:读写操作显式地分开。LL 不产生总线数据传送,这使得下面这段程序与前面采用交换操作、经过优化了的代码具有相同的特点(R1 中保存锁的地址):

```
lockit: LL      R2, 0(R1)        //load-linked
        BNEZ    R2, lockit       //若锁未被释放,则旋转
        DADDIU  R2, R0, #1       //置锁值为 1
        SC      R2, 0(R1)        //写入存储器
        BEQZ    R2, lockit       //如果存失败则转移
```

第一个分支形成环绕的循环体,第二个分支解决了两个处理器同时看到锁可用的情况下的争用问题。尽管旋转锁机制简单并且具有吸引力,但难以将它应用于处理器数量很多的情况,因为锁被释放时,处理器之间争用锁会产生大量的通信开销。

3. 同步性能问题

上面介绍的简单旋转锁不能很好地适应可扩放性。设想一下,如果大规模多处理机中所有的处理器都同时争用同一个锁,那么目录或总线作为所有处理器实现串行化的中心点,肯定是个瓶颈,会导致大量的争用和通信开销。

下面讨论如何用旋转锁来实现一个常用的高级同步原语——栅栏同步。栅栏强制所有到达该栅栏的进程进行等待,直到全部的进程到达栅栏,然后释放全部的进程,从而形成同步。栅栏的典型实现是用两个旋转锁:一个用来保护一个计数器,它记录已到达该栅栏的进程数;另一个用来封锁进程直至最后一个进程到达该栅栏。为了实现栅栏,一般要利用这样的功能:在一个变量上旋转等待直到它满足规定的条件。我们用 spin(condition)来表示这种情况。下面的程序是一种典型的实现,其中,lock 和 unlock 提供基本的旋转锁,变量 count 记录已到达栅栏的进程数,total 规定了要到达栅栏的进程总数,对 counterlock 加锁保证增量操作的原子性。release 用来封锁进程直到最后一个进程到达栅栏。spin(release==1)使进程等待直到全部的进程到达栅栏。

```
lock(counterlock);                 //确保更新的原子性
if(count == 0)release = 0;         //第一个进程则重置 release
count = count + 1;                 //到达进程数加 1
unlock(counterlock);               //释放锁
if(count == total){                //进程全部到达
    count = 0;                     //重置计数器
    release = 1;                   //释放进程
    }
```

```
    else {                              //还有进程未到达
        spin(release = 1);              //等待别的进程到达
        }
```

栅栏通常是在循环中使用，因此从栅栏释放出的进程在运行一段后又会再次到达该栅栏。假设其中有一个进程还没有离开栅栏，即停留在旋转等待操作上(例如操作系统重新调度进程后就可能发生这样的情况)。这时如果有个进程的执行比较快，又到达了栅栏，而上一次循环的进程中最后那个还没来得及离开该栅栏。那么这个“快”进程就会把 release 重新置为 0，从而把上次循环的“慢”进程“捆”在了这个栅栏上。这样所有的进程在这个栅栏的又一次使用中都会处于无限等待状态，因为已经到达该栅栏的进程数目总是达不到 total (上一次循环欠了 1 个)。

解决这个问题的一种方法是当进程离开栅栏时进行计数(和到达时一样)，在上次栅栏使用中的所有进程离开之前，不允许任何进程重用并初始化本栅栏。但这会明显增加栅栏的延迟和竞争。另一种解决办法是采用 sense_reversing 栅栏，每个进程均使用一个私有变量 local_sense，该变量初始化为 1。下面的程序给出了 sense_reversing 栅栏的代码，这种方法使用安全。但其性能仍旧比较差。对于 10 个处理器来说，当同时进行栅栏操作时，如果忽略对 Cache 的访问时间以及其他非同步操作所需的时间，则其总线事务数为 204 个，如果每个总线事务需要 100 个时钟周期，则总共需要 20 400 个时钟周期。

```
    local_sense = ! local_sense;        //local - sense 取反
    lock(counterlock);                  //确保更新的原子性
    count++;                            //到达进程数加 1
    unlock(counterlock);                //释放锁
    if(count == total){                 //进程全部到达
        count = 0;                      //重置计数器
        release = local_sense;          //释放进程
        }
    else {                              //还有进程未到达
        spin(release == local_sense);   //等待信号
        }
```

由上面这些例子可以看出，当多进程之间竞争激烈时，同步会成为瓶颈。当竞争很少而且同步操作也较少时，我们主要关心的是同步原语操作的延迟，即单个进程要花多长时间才完成一个同步操作。基本的旋转锁操作可在两个总线周期内完成：一个读锁，一个写锁。我们可以采用多种方法进行改进，使它在单个周期内完成操作。例如，可简单地在交换操作上旋转。如果锁经常处于未占用状态，这种方法很好；但如果锁已被占用，就将会导致大量的总线事务，因为每一个试图给锁变量加锁的操作均需要一个总线周期。不过，实际上旋转锁的延迟并不像上述例子中所示的那样糟糕，因为在实现中可以对 Cache 的写不命中加以优化。

上述例子中，更严重的问题在于进程进行同步操作的串行化。当有竞争时，串行化就会成为一个问题。因为它大幅度地增加了完成同步操作所需要的时间。栅栏的情况也差不多，甚至更严重。

10.2.5　同时多线程

线程是进程内的一个相对独立的、可独立调度和指派的执行单元。它只拥有在运行过程中必不可少的一点儿资源，如程序计数器、一组寄存器、堆栈等。所以线程切换时，只需保存和设置少量寄存器的内容，开销很小。线程切换只需要几个时钟周期，最快可以每个时钟周期切换一次。而进程的切换一般需要成百上千个处理器时钟周期。

实现多线程的主要方法有以下两种。

(1) 细粒度多线程。它在每条指令之间都能进行线程的切换，从而使得多个线程可以交替执行。通常以时间片轮转的方法实现这样的交替执行，在轮转的过程中跳过处于停顿的线程。为了使细粒度多线程能达到实用的水平，CPU必须在每个时钟周期都能进行线程的切换。

优点：不仅能够隐藏由长时间停顿引起的吞吐率的损失，而且能够隐藏由短时间停顿带来的损失。

缺点：减慢了单个线程的执行，这是因为即使没有任何停顿的线程也不能连续执行，而是会因其他线程的指令的插入执行而被延迟。

(2) 粗粒度多线程。它是针对细粒度多线程的缺点而提出的。粗粒度多线程之间的切换只发生在时间较长的停顿出现的时候。这一改变使得粗粒度多线程不需要像细粒度那样自由地切换，减少了切换次数，并且也不太会降低单个线程的执行速度。

缺点：减少吞吐率损失的能力有限，特别是对于较短的停顿来说更是如此。由于实现粗粒度多线程的CPU只能执行单个线程的指令，不能交叉执行多个线程，因此当发生停顿时，流水线必须排空或暂停。新线程也有个填满流水线的过程，填满后才能不断地流出指令执行结果。由于有启动开销，粗粒度多线程对于减少长时间停顿所带来的损失较为有效，此时流水线的建立时间相对于停顿时间可以忽略不计。

1. 将线程级并行转换为指令级并行

同时多线程技术(Simultaneous MultiThreading，SMT)是一种在多流出、动态调度的处理器上同时开发线程级并行和指令级并行的技术，它是多线程技术的一种改进。提出SMT的主要原因是现代多流出处理器通常含有多个并行的功能单元，而单个线程不能有效地利用这些功能单元。而且，通过寄存器重命名和动态调度机制，来自各个独立线程的多条指令可以同时流出，而不用考虑它们之间的相互依赖关系，其相互依赖关系将通过动态调度机制得以解决。

在**粗粒度多线程**超标量处理器中，通过线程的切换部分隐藏了长时间停顿带来的开销，提高了硬件资源的利用率。尽管这样减少了完全空闲的时钟周期，但由于ILP有限，而且在粗粒度多线程处理器中，只有发生停顿时才进行线程切换，而且新线程还有个启动期，所以仍然可能有一些完全空闲的时钟周期。

在**细粒度多线程**超标量处理器中，线程的交替执行消除了完全空闲的时钟周期。由于在每个时钟周期内只能流出一个线程的指令，ILP的限制导致了一些时钟周期中依然存在不少空闲流出槽。

在SMT超标量处理器中，是同时开发实现线程级并行和指令级并行，允许在同一个时钟周期中由多个线程共同使用流出槽。理想情况下，流出槽的利用率只受限于多个线程对资源的需求和可用资源间的不平衡，但实际中也受其他一些因素的限制，包括活跃线程的个数、缓冲大小的限制、从多个线程取出足够多条指令的能力、线程之间哪些指令组合可以同时流出等。

如上所述，动态调度的处理器已经具备了开发线程级并行所需的许多硬件设置。具体来说，动态调度超标量处理器有一组很多的虚拟寄存器，可以用来保存各独立线程的寄存器组（假设每个线程都有一个独立的重命名表）。由于寄存器重命名机制给各寄存器提供了唯一的标识，多个线程的指令可以在数据路径上混合执行，而不会导致各线程之间源操作数和目的操作数的混乱。这表明，只要为每个线程设置重命名表、分别设置各自的程序计数器并为多个线程提供指令确认的能力，则多线程可以在一个乱序执行的处理器的基础上实现。

2. 同时多线程处理器的设计

多个线程的混合执行不可避免地会影响单个线程的执行速度，类似的问题在取指阶段也存在。为提高单个线程的性能，应该为指定的优先线程尽可能多地向前取指（或许在分支指令的两条路径上都要向前取指），并且在分支预测失败和预取缓冲器不命中的情况下清空取指单元。但是这样限制了其他线程可用来调度的指令条数，从而降低了吞吐率。所有的多线程处理器都必须在这里寻求一种折中方案。

实际上，进行资源划分以及在单个线程性能和多个线程性能之间进行平衡并不会像我们想象的那么复杂，至少对于目前的超标量处理器来说是这样。例如，对于现在的每个时钟周期流出4～8条指令的计算机来说，有几个活跃线程就够了，优先线程就更少了。只要一有可能，处理器就运行指定的优先线程。从取指阶段开始就优先处理优先线程：只要优先线程的指令预取缓冲区未满，就为它们优先取指。只有当优先线程的缓冲区填满以后才为其他线程预取指令。需要注意的是，当有两个优先线程时，意味着需要并发预取两条指令流，这给取指部件和指令Cache都增添了复杂度。同样地，指令流出单元也要优先考虑指定的优先线程，只有当优先线程停顿不能流出的时候才考虑其他线程。

下面是设计同时多线程处理器时面临的其他主要问题。

（1）需要设置更大的寄存器组，用来保存多个线程的现场。

（2）不能影响时钟周期，特别是在关键路径上，如指令流出和指令完成。指令流出时，有更多的候选指令需要考虑；在指令完成时，选择提交哪些指令可能会比较困难。

（3）需要保证由于并发执行多个线程带来的Cache冲突和TLB冲突不会导致明显的性能下降。

在考虑这些问题时，需要重视以下两个实际情况：第一，在许多情况下，多线程所导致的潜在额外性能开销是很小的，简单的线程切换选择算法就足够好了；第二，目前的超标量处理器的效率是比较低的，还有很大的改进余地，即使增加一些开销也是值得的。

由于同时多线程在多流出超标量处理器上开发线程级并行，所以最适合于应用到面向服务器市场的高端处理器上。另外，还可以限定多线程的并发数量，这样就可以最大限度地提高单个线程的性能。

3. 同时多线程的性能

一个具体实例的模拟结果表明，在超标量处理器上增添8个线程的同时多线程所获得的吞吐率的提高很显著，达1.7～4.2倍，平均3倍。同时多线程处理器的取指部件和功能部件的利用率大为提高。分支预测的精确度和指令Cache的命中率的比较同样令人惊讶：同时多线程处理器的性能更好。

超标量处理器本身功能十分强大，它具有很大的一级Cache、二级Cache以及大量的功能部件。仅采用指令级并行，不可能利用全部的硬件性能，因此超标量处理器的设计者不可能不考虑使用诸如同时多线程这样的技术来开发线程级并行。将超标量和同时多线程结合起来，在指令级并行基础上进一步开发线程级并行，可以获得显著的性能提高。

10.2.6　大规模并行处理机MPP

1. 并行计算机系统结构

目前流行的高性能并行计算机系统结构通常可以分成以下5类。

(1) 并行向量处理机(Parallel Vector Processor，PVP)；

(2) 对称式共享存储器多处理机(Symmetric shared memory MultiProcessor，SMP)；

(3) 分布式共享存储器多处理机(Distributed Shared memory MultiProcessor，DSM)；

(4) 大规模并行处理机(Massively Parallel Processor，MPP)；

(5) 机群计算机(Cluster)。

1) 并行向量处理机

PVP系统一般由若干台高性能向量处理机(VP)构成。这些向量处理机是专门设计和定制的，拥有很高的向量处理性能。PVP中经常采用专门设计的高带宽的交叉开关网络，把各VP与共享存储器模块SM连接起来。这样的机器通常不使用Cache，而是使用大量的向量寄存器和指令缓冲器。

2) 对称式共享存储器多处理机和分布式共享存储器多处理机

这两种体系结构在上面已经详细讨论过，这里不再重复。

3) 大规模并行处理机

MPP往往是超大规模的计算机系统。它具有以下特点。

(1) 处理结点使用商用微处理器，而且每个结点可以有多个微处理器；

(2) 具有较好的可扩放性，能扩展成具有成百上千个处理器；

(3) 系统中采用分布非共享的存储器，各结点有自己的地址空间；

(4) 采用专门设计和定制的高性能互连网络；

(5) 采用消息传递的通信机制。

4) 机群计算机

机群是一种价格低廉、易于构建、可扩放性极强的并行计算机系统。它由多台同构或异构的独立计算机通过高性能网络或局域网互连在一起，协同完成特定的并行计算任务。第12章将专门讨论机群。

2. 大规模并行处理机 MPP

1）MPP 的出现和发展

略。

2）MPP 系统概述

MPP 结构的一个重要特性是可扩放性，不仅处理器的数量可扩放至数千个处理器，而且主存、I/O 能力和带宽也能随处理器数量的增长而成比例地增长。MPP 主要采用了以下技术来提高系统的可扩放性。

（1）使用物理上分布的主存体系结构，使分布式主存的总容量和总带宽能随处理结点数量的增加而增加。这种分布式主存结构比集中式主存结构具有潜在的更高可扩放性。

（2）处理能力、主存与 I/O 能力平衡发展。随着处理结点数量的增长，不仅 MPP 系统的处理能力随之增长，而且系统的主存与 I/O 能力也随之平衡增长。如果系统的主存与 I/O 能力不能随处理能力的提高而平衡发展，那么，高速的处理能力就毫无价值。

（3）计算能力与并行性平衡发展。一个计算作业可分解成多个任务分配到多个处理结点上并行执行。并行处理的性能在很大程度上依赖于计算/通信比率，如果这个比值较小，就说明并行进程/线程管理及通信同步的时间开销将占有作业执行时间的较大部分。因此，MPP 的计算能力随处理结点数量的增长而增长时，也要使处理结点的并行能力平衡发展。

Intel ASCI 系统遵循了小结点、紧耦合网络互连和计算结点的微内核操作系统，是一种更传统的 MPP 方法。它是 Intel Paragon MPP 系统的后代。SP2 和 Intel ASCI 都是使用 NORMA 访存模型的消息传递多计算机，结点间通信依靠机器中的显式消息传递。

SGI/CrayOrigion 2000 代表一种构造 MPP 的不同方法，其特征为一个可全局存取的物理上分布的主存系统，使用硬件支持 Cache 的一致性。

10.2.7 多处理机实例 1：T1

在近年来推出的处理器中，T1 是独具特色的一个，它把重点放在了开发线程级并行性，而不是指令级并行性。它在一块芯片上集成了 8 个处理器核，以期对于服务器应用程序实现高吞吐率。与之不同，双核的 Power5、Opteron、Pentium D 则是采用多流出和多核技术。开发大量 ILP 并行性需要用复杂得多的处理器。

除了是重点开发 ILP 还是 TLP 的区别外，这些多核处理器还有一些根本的不同，包括：

（1）它们在对浮点运算提供的支持以及浮点运算的性能上有很大的不同。Power5 主要强调浮点运算的性能，Opteron 和 Pentium 也为浮点运算分配了大量的资源，它们都包含大量的浮点运算硬件。而 T1 则几乎忽略了这方面。它们的侧重点是不同的。因此，用只包含整数运算的测试程序来比较这几个处理器的性能是不合理的，同样地，用只包含浮点运算的测试程序对 T1 来说也是不合理的。

（2）它们的多处理器扩展能力不同，这对存储器的设计以及外部接口的使用有很大的影响。Power5 的可扩展性是最好的，Pentium 和 Opteron 也对多处理器扩展提供了一定的支持，但 T1 则不支持扩展成更大的系统。

（3）所用的实现技术差别很大，难以对它们的晶片大小和功耗进行比较。

(4) 对存储器系统及其带宽的要求不同。对于像TPC-C这类Cache不命中率高的测试程序来说,访存带宽高的处理器占有很大优势。

另外,还有一些模拟结果表明,对于SPECJBB05和类TPC-C测试程序来说,T1的每瓦性能要比其他处理器的高很多。这至少说明对于多线程程序来说,在每瓦性能方面,采用TLP方法比ILP方法要有效得多。显然,现在要对以开发TLP为主的方法能否在商业上获得成功做出结论还太早。但如果服务器应用程序中有足够的线程来使得T1充分忙碌,而且每线程的性能是可以接受的,那么T1采用的方法就很难被超越。如果在服务器或桌面环境中单线程的性能仍然很重要,我们将会看到市场的进一步分化,会出现各种面向完全不同环境的处理器,包括追求高吞吐率的环境和追求单一线程高性能的环境等。

10.2.8 多处理机实例2:Origin 2000

SGI公司将Cray Research子公司的开关网络技术与SMP系统的优点结合起来,推出了Origin 2000系列可扩展服务器产品。该系列包括Origin 200、Origin 2000 Deskside、Origin 2000 Rack和Cray Origin 2000共4种机器。Origin 200服务器是入门级的系统,具有中等扩充能力,最多可以达到4个处理器。Origin 2000 Deskside桌面服务器系统支持的处理器数目最多为8个,Origin 2000 Rack机柜服务器系统支持的处理器数目最多为16个,Cray Origin 2000服务器系统具有大规模扩充能力,支持的处理器数目最多可达到128个。

Origin 2000系列服务器产品不仅具有SMP的易编程和平稳扩充特性,而且还具有MPP的高可扩放性,应用非常广泛。该系列服务器综合平衡了高性能、可扩放性、可用性和兼容性,能满足许多应用的需求,例如,可作为企业、商业金融机构以及政府机构的信息管理服务器,可用于Web服务、数据仓库、可视化服务、科学计算、图像处理和仿真等。Origin 2000服务器系列的I/O带宽可达102GB/s,系统传输速率比同类SMP服务器快几十倍,是处理、存储和传输各种多媒体信息的理想系统。

习　　题

1. 概念题

【题10.1】 解释以下名词

集中式共享多处理机	分布式共享多处理机	计算/通信比	多Cache一致性
写作废协议	写更新协议	栅栏同步	旋转锁
同时多线程	细粒度多线程技术	粗粒度多线程技术	
SMP	MPP		

2. 填空题

【题10.2】 能实现指令、程序、任务级并行的计算机系统是________。

【题 10.3】 多处理机结构由若干台独立的计算机组成，每台计算机能够独立执行自己的________，Flynn 称这种结构为________结构。

【题 10.4】 现有的 MIMD 机器可分为________和________两类。每一类代表了一种存储器的结构和互连策略。

【题 10.5】 消息传递机器根据简单的________协议，通过________来请求服务或传送数据。

【题 10.6】 对应于两种地址空间的组织方案，分别有________和________两种通信机制。

【题 10.7】 对称式共享存储器系统结构一般都支持对共享数据和私有数据的 Cache 缓存。私有数据是指________，而共享数据则是指________。

【题 10.8】 多处理机的 Cache 一致性问题是指________。

【题 10.9】 实现 Cache 一致性协议的关键是________。

【题 10.10】 采用目录协议解决多处理机的 Cache 一致性问题时，根据目录的结构，可以把目录协议分成________、________和________三类。其中，具有可扩展性的目录协议是________和________。

【题 10.11】 链式目录有________和________两种实现方法。

【题 10.12】 在基于总线互连的多处理机中，采用________协议来解决 Cache 一致性问题。

【题 10.13】 实现 Cache 一致性的协议有________和________两类。

【题 10.14】 T1 是作为服务器处理器发布的多核多处理器。每个 T1 处理器中有________个处理器核，每个核最多支持________个线程。每个处理器核中都有一条________段的单流出流水线。

【题 10.15】 Origin 2000 的关键技术包括________技术和________。

3. 问答题

【题 10.16】 并行处理面临着哪两个重要的挑战？分别采用什么方法解决？

【题 10.17】 共享存储器通信主要有哪些优点？

【题 10.18】 消息传递通信机制主要有哪些优点？

【题 10.19】 在分布式多处理机中将存储器分布到各结点有什么好处？

【题 10.20】 大规模的多处理机中，存储器在物理上是分布于各个处理结点中。在逻辑地址空间的组织方式以及各个处理器之间通信的实现上，有哪两种地址空间的组织方案？

【题 10.21】 简述共享存储器通信机制的主要优点。

【题 10.22】 简述消息传递通信机制的主要优点。

【题 10.23】 在实现消息传递机制的硬件上怎样支持共享存储器通信机制？

【题 10.24】 简述多处理机中产生 Cache 一致性问题的原因。

【题 10.25】 一致的存储系统应满足哪三点？

【题 10.26】 什么是共享数据的迁移和复制？这么做的原因是什么？

【题 10.27】 什么是多处理机的 Cache 一致性协议？给出解决一致性的监听协议和目录协议的工作原理。

【题 10.28】 简述写更新协议和写作废协议性能上的差别。

【题 10.29】 简述细粒度多线程技术与粗粒度多线程技术的优缺点。

【题 10.30】 目前流行的高性能并行计算机系统结构通常可以分成哪5类?

【题 10.31】 简述超大规模计算机系统 MPP 的特点。

4. 应用题

【题 10.32】 一个具有32台处理机的系统,对远程存储器访问时间是2000ns。除了通信以外,假设计算中的访问均命中局部存储器。当发出一个远程请求时,本地处理机挂起。处理机的时钟周期时间是10ns,假设指令基本的CPI为1.0(设所有访存均命中Cache)。对于下述两种情况:

(1) 没有远程访问;

(2) 0.5%的指令需要远程访问。

试问前者比后者快多少?

【题 10.33】 在基于总线的小型多处理器系统上,将基于监听的Cache一致性协议应用于写直达Cache,采用不按写分配,采用写作废协议。画出Cache块的状态转换图。

【题 10.34】 在标准的栅栏同步中,设单个处理器的通过时间(包括更新计数和释放锁)为 C,求 N 个处理器一起进行一次同步所需要的时间。

【题 10.35】 采用排队锁和 fetch and increment 重新实现栅栏同步,并将它们分别与采用旋转锁实现的栅栏同步进行性能比较。

【题 10.36】 假设某条总线上有10个处理器同时准备对同一变量加锁。如果每个总线事务处理(读不命中或写不命中)的时间是100个时钟周期,而且忽略对已调入Cache中的锁进行读写的时间以及占用该锁的时间。

(1) 假设该锁在时间为0时被释放,并且所有处理器都在旋转等待该锁。问:所有10个处理器都获得该锁所需的总线事务数目是多少?

(2) 假设总线是非常公平的,在处理新请求之前,要先全部处理好已有的请求。并且各处理器的速度相同。问:处理10个请求大概需要多少时间?

【题 10.37】 有些机器实现了专门的锁广播一致性协议,实现上可能使用不同的总线。假设使用写广播协议,重新给出题10.36旋转锁的时间计算。

题 解

1. 概念题

【题 10.1】 解释以下名词

集中式共享多处理机 —— 也称为对称式共享存储器多处理机SMP。它一般由几十个处理器构成,各处理器共享一个集中式的物理存储器,这个存储器相对于各处理器的关系是对称的。

分布式共享多处理机 —— 它的共享存储器分布在各台处理机中,每台处理机都带有自

己的本地存储器，组成一个“处理机-存储器”单元。但是这些分布在各台处理机中的实际存储器又合在一起统一编址，在逻辑上组成一个共享存储器。这些处理机存储器单元通过互连网络连接在一起，每台处理机除了能访问本地存储器外，还能通过互连网络直接访问在其他处理机存储器单元中的“远程存储器”。

计算/通信比——反映并行程序性能的一个重要的度量。在并行计算中，它是指每次数据通信要进行的计算与通信开销的比值。

多 Cache 一致性——多处理机中，当共享数据进入 Cache，就可能出现多个处理器的 Cache 中都有同一存储器块的副本，当其中某个处理器对其 Cache 中的数据进行修改后，就会使得其 Cache 中的数据与其他 Cache 中的数据不一致。

写作废协议——在处理器对某个数据项进行写入之前，它需拥有对该数据项的唯一的访问权。这是通过作废其他 Cache 中的副本来实现。

写更新协议——当一个处理器对某数据项进行写入时，它把该新数据广播给所有其他 Cache。这些 Cache 用该新数据对其中的副本进行更新。

栅栏同步——栅栏强制所有到达该栅栏的进程进行等待。直到全部的进程到达栅栏，然后释放全部进程，从而形成同步。

旋转锁——处理机环绕一个锁不停地旋转而请求获得该锁。

同时多线程——是一种在多流出、动态调度的处理器上同时开发线程级并行和指令级并行的技术，它是多线程技术的一种改进。

细粒度多线程技术——是一种实现多线程的技术。它在每条指令之间都能进行线程的切换，从而使得多个线程可以交替执行。通常以时间片轮转的方法实现这样的交替执行，在轮转的过程中跳过处于停顿的线程。

粗粒度多线程技术——是一种实现多线程的技术。只有线程发生较长时间的停顿时才切换到其他线程。

SMP——对称式共享存储器多处理机。

MPP——即大规模并行处理，按照当前的标准，具有几百台至几千台处理机的任何机器都是大规模并行处理系统。

2. 填空题

【题 10.2】 答：MIMD

【题 10.3】 答：程序、MIMD

【题 10.4】 答：集中式共享存储器结构、分布式存储器结构

【题 10.5】 答：网络、传递消息

【题 10.6】 答：共享存储器、消息传递

【题 10.7】 答：只供一个处理器使用的数据、供多个处理器共同使用的数据

【题 10.8】 答：不同处理机的 Cache 中同一地址的数据块可能不一致，Cache 与共享主存中同一地址的数据块也可能不一致

【题 10.9】 答：跟踪共享数据块的状态

【题 10.10】 答：全映像目录、有限映像目录、链式目录、有限映像目录、链式目录

【题 10.11】 答：单链法、双链法

【题10.12】 答：总线监听

【题10.13】 答：目录式协议、监听式协议

【题10.14】 答：8、4、6

【题10.15】 答：CrayLink开关网络、Cellular IRIX操作系统

3. 问答题

【题10.16】 答：①程序中有限的并行性。通过采用并行性更好的算法来解决。②相对较高的通信开销。减少远程访问延迟主要靠系统结构支持和编程技术。

【题10.17】 答：①与常用的集中式多处理机使用的通信机制兼容。②当处理器通信方式复杂或程序执行动态变化时易于编程，同时在简化编译器设计方面也占有优势。③当通信数据较少时，通信开销较低，带宽利用较好。④通过硬件控制的Cache减少了远程通信的频度，减少了通信延迟以及对共享数据的访问冲突。

【题10.18】 答：①硬件较简单。②通信是显式的，从而引起编程者和编译程序的注意，着重处理开销大的通信。

【题10.19】 答：①如果大多数的访问是针对本结点的局部存储器，则可降低对存储器和互连网络的带宽要求；②对局部存储器的访问延迟较小。

【题10.20】 答：①第一种方案：物理上分离的多个存储器作为一个逻辑上共享的存储空间进行编址。处理器之间是通过用load和store指令对相同存储器地址进行读/写操作来实现的。②第二种方案：把每个结点中的存储器编址为一个独立的地址空间，不同结点中的地址空间之间是相互独立的。即整个地址空间由多个独立的地址空间构成，它们在逻辑上也是独立的，远程的处理器不能对其直接寻址。各处理机之间采用消息传递通信机制进行通信，数据通信要通过在处理器之间显式地传递消息来完成。

【题10.21】 答：(1) 与常用的对称式多处理机使用的通信机制兼容。

(2) 当处理器之间通信方式复杂或在执行过程中动态变化时，采用共享存储器通信，编程容易，同时在简化编译器设计方面也占有优势。

(3) 采用大家所熟悉的共享存储器模型开发应用程序，而把重点放到解决对性能影响较大的数据访问上。

(4) 当通信数据量较小时，通信开销较低，带宽利用较好。

(5) 可以通过采用Cache技术来减少远程通信的频度。这是通过对所有数据(包括共享的和私有的)进行Cache缓冲来实现的。

【题10.22】 答：(1) 硬件更简单。特别是在与可扩放共享存储器实现方案相比时更是如此。

(2) 通信是显式的，因此更容易搞清楚何时发生通信以及通信开销是多少。

(3) 显式通信可以让编程者重点注意并行计算的主要通信开销，使之有可能开发出结构更好、性能更高的并行程序。

(4) 同步很自然地与发送消息相关联，能减少不当的同步带来错误的可能性。

【题10.23】 答：所有对共享存储器的访问均要求操作系统提供地址转换和存储保护功能，即将存储器访问转换为消息的发送和接收。

【题10.24】 答：①Cache的引进对I/O操作产生了一致性问题，因为Cache中的内容

可能与由I/O子系统输入输出形成的存储器对应部分的内容不同。②对共享数据，不同处理器的Cache都保存有对应存储器单元的内容，因而在操作中就可能产生数据的不一致。

【题10.25】　答：(1)处理器P在对存储单元X进行一次写之后又对X进行读，在这读和写之间没有其他处理器对X进行写，则P读到的值总是刚写进去的值。

(2) 处理器P对存储单元X进行写之后，另一处理器Q对X进行读，在这读和写之间没有其他对X的写，则Q读到的值应为P写进去的值。

(3) 对同一存储单元的写是串行化的。即任意两个处理器对同一存储单元的两次写，从各个处理器的角度看来顺序都是相同的。

【题10.26】　答：共享数据的迁移是把远程的共享数据复制一份，迁入本地Cache供本处理器使用。这是为了减少对远程共享数据的访问延迟，减少对共享存储器带宽的要求。

共享数据的复制则是把多个处理器需要同时读取的共享数据在这些处理器的本地Cache中各存放一个副本。复制不仅可以减少访问共享数据的延迟，还可减少访问共享数据所产生的冲突。

【题10.27】　答：(1) 对多个处理器维护一致性的协议称为多Cache一致性协议。

(2) 目录协议的工作原理：采用一个集中的数据结构——目录。对于存储器中的每一个可以调入Cache的数据块，在目录中设置一条目录项，用于记录该块的状态以及哪些Cache中有副本等相关信息。目录协议根据该项目中的信息以及当前要进行的访问操作，依次对相应的Cache发送控制消息，并完成对目录项信息的修改。此外，还要向请求处理器发送响应信息。

(3) 监听协议的工作原理：每个Cache除了包含物理存储器中块的数据副本之外，也保存着各个块的共享状态信息。Cache通常连在共享存储器的总线上，当某个Cache需要访问存储器时，它会把请求放到总线上广播出去，其他各个Cache控制器通过监听总线来判断它们是否有总线上请求的数据块。如果有，就进行相应的操作。

【题10.28】　答：(1) 对同一数据的多个写而中间无读操作的情况，写更新协议需进行多次写广播操作，而在写作废协议下只需一次作废操作。

(2) 对同一块中多个字进行写，写更新协议对每个字的写均要进行一次广播，而在写作废协议下仅在对本块第一次写时进行作废操作即可。写作废是针对Cache块进行操作，而写更新则是针对字(或字节)进行操作。

(3) 从一个处理器写到另一个处理器读之间的延迟通常在写更新模式中较低，因为它写数据时马上更新了相应的其他Cache中的内容(假设读的处理器Cache中有此数据)。而在写作废协议中，需要读一个新的副本。

【题10.29】　答：(1) 细粒度多线程技术的主要优点是不仅能够隐藏由长时间停顿引起的吞吐率的损失，而且能够隐藏由短时间停顿带来的损失。其主要的缺点是减慢了单个线程的执行，这是因为即使没有任何停顿的线程也不能连续执行，而是会因其他线程的指令的插入执行而被延迟。

(2) 粗粒度多线程技术减少了线程切换次数，并且也不太会降低单个线程的执行速度，但它减少吞吐率损失的能力有限，特别是对于较短的停顿来说更是如此。

【题10.30】　答：

(1) 并行向量处理机PVP；

(2) 对称式共享存储器多处理机 SMP；

(3) 分布式共享存储器多处理机 DSM；

(4) 大规模并行处理机 MPP；

(5) 机群计算机 Cluster。

【题 10.31】 答：

(1) 处理结点使用商用微处理器，而且每个结点可以有多个微处理器；

(2) 具有较好的可扩放性，能扩展成具有成百上千个处理器；

(3) 系统中采用分布非共享的存储器，各结点有自己的地址空间；

(4) 采用专门设计和定制的高性能互连网络；

(5) 采用消息传递的通信机制。

4. 应用题

【题 10.32】

解：已知远程访问率 $p=0.5\%$，远程访问时间 $t=2000\text{ns}$，时钟周期 $T=10\text{ns}$

远程访问开销 $C=t/T=2000\text{ns}/10\text{ns}=200$(时钟周期数)

有 0.5%远程访问的机器的实际 CPI_2 为：

$$CPI_2=CPI_1+p\times C=1.0+0.5\%\times 200=2.0$$

只有局部访问的机器的基本 $CPI_1=1.0$

$$CPI_2/\ CPI_1=2.0/1.0=2(\text{倍})$$

因此，没有远程访问状态下的机器速度是有 0.5%远程访问的机器速度的 2 倍。

【题 10.33】

解：写作废协议中(采用写直达法)，Cache 块的状态转换图：响应来自 CPU 的请求(图 10.1)。

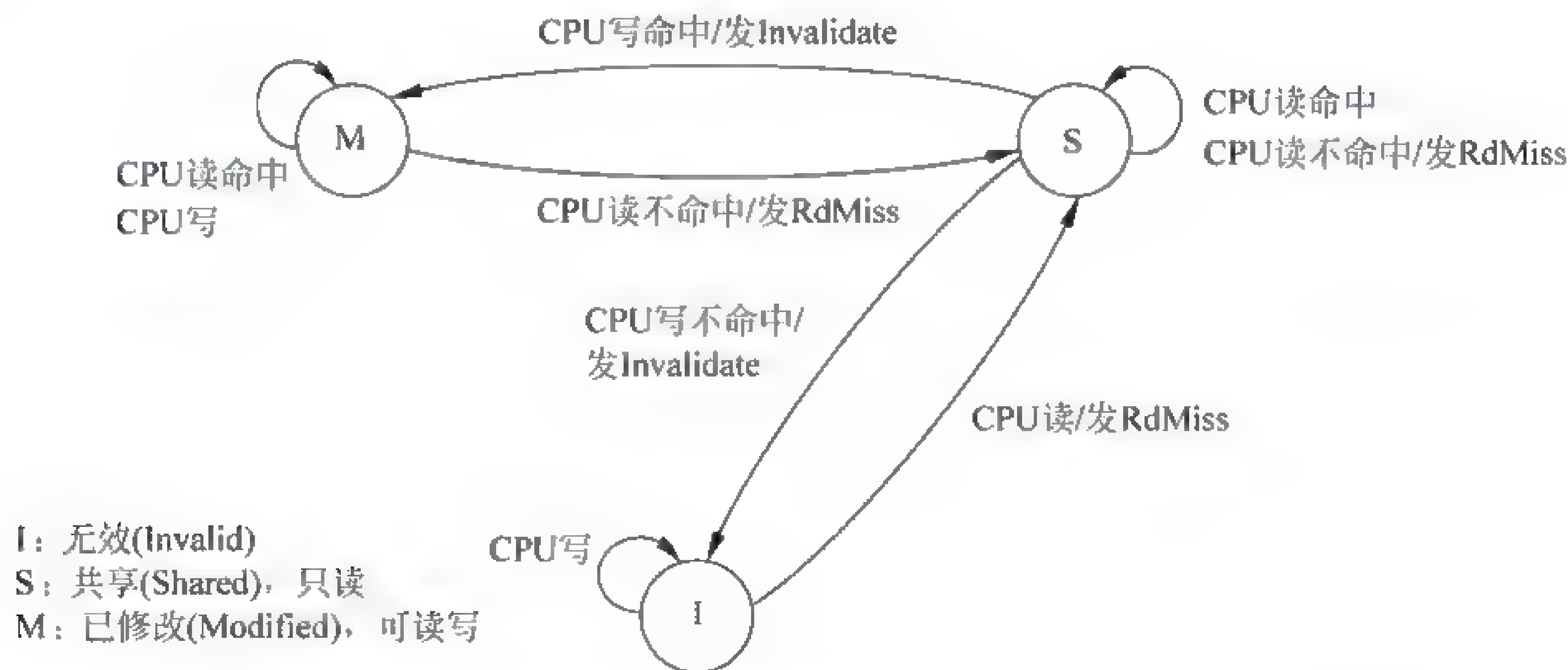


图 10.1　Cache 块的状态转换图：响应来自 CPU 的请求

写作废协议中(采用写直达法)，Cache 块的状态转换图：响应来自总线的请求(图 10.2)。

【题 10.34】

解：我们忽略读写锁的时间。N 个处理器中的每一个都需要 C 个时钟周期来锁住与栅栏相关的计数器，修改它的值，然后释放锁。考虑最坏情况，所有 N 个处理器都要对计数器

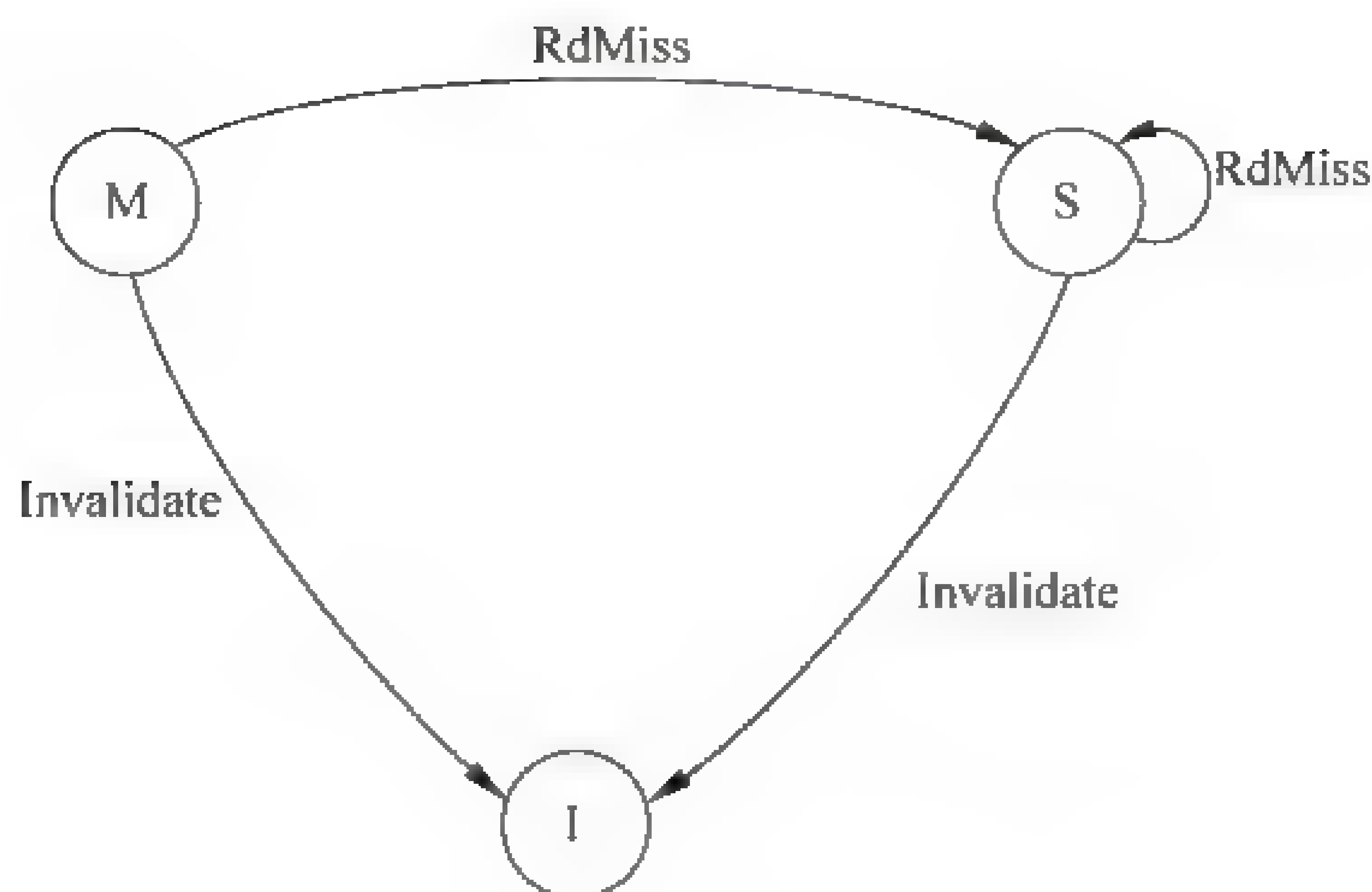


图 10.2 Cache 块的状态转换图：响应来自总线的请求

加锁并修改它的值，由于锁只能顺序访问计数器，在同一时间，只能有一个处理器修改计数器的数据。所以，总共要花 NC 个时钟周期使得所有的处理器都到达数据栅栏。

【题 10.35】

解：

```
fetch - and - increment(count);
    if (count = total){                 //进程全部到达
        count = 0;                      //重置计数器
        release = 1;                    //释放进程
        }
    else{                               //还有进程未到达
        spin(release = 1);              //等待信号
        }
```

当有 N 个处理器时，上述代码执行 fetch and increment 操作 N 次，当访问释放操作的时候，有 N 个 Cache 未命中。当最后一个处理器到达栅栏条件后，release 被置为"1"，此时有 $N-1$ 个 Cache 未命中(对于最后一个到达栅栏的处理器，当它读 release 的时候，将在主存中命中)。所以，共有 $3N-1$ 次总线传输操作。如果有 10 个处理器，则共有 29 次总线传输操作，总共需要 2900 个时钟周期。

【题 10.36】

解：当 i 个处理器争用锁的时候，它们都各自完成以下操作序列，每一个操作产生一个总线事务。

(1) 访问该锁的 i 个 LL 指令操作；

(2) 试图占用该锁(并上锁)的 i 个 SC 指令操作；

(3) 一个释放锁的存操作指令。

因此对于 i 个处理器来说，一个处理器获得该锁所要进行的总线事务的个数为 $2i+1$。这里假设关键代码段的执行时间可以忽略不计。

假设一共有 n 个处理器。在最开始时，共有 n 个处理器在争用该锁，一个处理器胜出，完成执行后释放该锁，其总线事务的个数为 $2n+1$；接下来，剩下的 $n-1$ 个处理器继续争用该锁，其总线事务的个数为 $2(n-1)+1$；其余以此类推。由此可知，总的总线事务个数为：

$$\sum_{i=1}^{n}(2i+1)=n(n+1)+n=n^2+2n$$

对于10个处理器来说,其总线事务数为120个,需要12 000个时钟周期。

【题10.37】

解:当实现了专门的锁广播一致性协议后,每当一把锁被释放的时候,和锁相关的值将被广播到所有处理器,这意味着在处理器对锁变量进行读操作的时候,未命中的情况永远不会发生。

假定每个Cache都有一个数据块保留锁变量的初值。通过表10.1可以知道,10次上锁/释放锁的平均时间是550个时钟周期,总时间是5500个时钟周期。

表10.1　旋转锁的时间计算

事　件	持续时间
所有处理器都读(命中)锁	0
释放锁的处理器进行写(不命中)广播	100
读(命中)锁(处理器认为锁是空闲的)	0
一个处理器进行写交换广播,同时还有9个写广播	1000
一个处理器得到并释放锁的总时间	1100

第11章　多核架构与编程

11.1　基本要求与难点

11.1.1　基本要求

(1) 掌握对多核架构的需求。
(2) 掌握多核组织结构以及 Intel x86 和 ARM11 MPCore 多核架构。
(3) 掌握并行编程模型、并行语言和并行算法。
(4) 理解几个多核编程实例。

11.1.2　难点

(1) 为什么要采用多核?
(2) 多核架构。
(3) 面向多核的编程方法及实例。

11.2　知识要点

多核处理器又称**芯片多处理器**(Chip Multiprocessor,CMP),是指在单个芯片内集成两个或多个处理器。其中,芯片内的每个处理器称作“核”,包含一套全部独立的处理器部件,如寄存器、ALU、流水线硬件、指令 Cache 和数据 Cache 等。除了多个核之外,现代多核芯片还包含共享的或独立的第二级(L2)Cache 甚至第三级(L3)Cache。

11.2.1　多核架构的需求

自从电子计算机诞生以来,微处理器系统性能上的增长主要来源于两个方面:一是时钟频率的提高;二是处理器芯片体系结构的改进。其中,体系结构的改进主要体现在并行度的不断增加方面。但在单核处理器系统中,无论是时钟频率,还是并行度都已接近了极限,很难再进一步地提高了,这时就需要采用更为先进的多核技术。

多核技术的好处非常明显。首先,由于计算系统拥有多个执行内核,可以同时进行并行运算,因此可以显著提升系统的计算能力,同时每个内核的主频可以比以前低,因而系统的

总体功耗增加不大。其次,与多CPU技术相比,多核处理器采用了与单CPU相同的硬件体系结构,用户在提升计算能力的同时无须进行任何硬件上的改变。然而,要真正发挥多核的潜能却并不容易,因为针对多核或多线程的软件开发要比单核或单线程编程工作艰难得多。而今,随着多核的普及,如何开发与多核相适应的软件日益成为研究人员所关注的热点问题。

1. 功耗与散热问题

随着芯片密度和时钟频率的不断提高,系统的功耗呈现出指数性增长的趋势。功耗的过快增长,一方面会增加用户的使用成本,另一方面还会给系统设计工程师带来诸如散热等诸多难题。CPU的发热量主要取决于处理器的密度和时钟频率这两个主要因素,与两者是正比关系。

在传统的体系结构中,每一代处理器所产生的热量增加率都要大于时钟频率的增加率。因此,在一些高端处理器的应用中,采用风扇散热的方式已经不能满足系统的要求了,取而代之的是液体散热方法,如水冷、液氮等技术。所以,受功率和散热方法的限制,时钟频率不能无限制地增加。

另一方面,控制功率密度(芯片密度)也是一种有效的降低功耗的方法,而控制芯片密度的一种主要方法就是使用更大面积的Cache存储芯片。Cache的晶体管相对较小,功率密度也比逻辑电路的要小。为了降低系统功耗,Cache占整个芯片面积的百分比比重越来越大,随着芯片内晶体管密度的增加,其面积逐渐超过了50%。

2. 并行度问题

处理器设计中,组织的变化主要集中在增加指令级并行度上,以便使处理器每个时钟能做更多的工作。按照时间的顺序,这些变化主要包括流水线技术、超标量技术和同时多线程技术等。就流水线技术而言,流水段越来越多;但是,在实际的实现中,肯定会存在着流水段上限。因为流水段越多,逻辑电路、互连结构以及控制信号就越复杂。就超标量技术而言,超标量组织也是通过增加并行流水线的个数来提高性能的。同样,随着流水线个数的增加,需要更复杂的逻辑管理冲突和调度指令使用资源,因此所获取的收益越来越小。冲突和资源依赖往往会造成多流水线不能充分利用,甚至单个线程就能让并行流水线饱和。就SMT技术而言,线程在一组流水线上调度的复杂度也往往会限制线程的个数和可有效利用的流水线的个数,因此性能的改进也是有限的。

随着系统结构复杂度的进一步提高,处理器芯片的设计和制造也会存在一些问题。对于超长流水线、超标量流水线和多寄存器体的SMT而言,逻辑电路复杂度的增加意味着控制和信号传送所占面积的增加,导致芯片设计、制造和调试难度的增加。因此,受控制逻辑复杂度的限制,流水线的宽度和深度都是有限的。因此,要想进一步增加并行度,又不增加控制的复杂度,只能选择增加处理器的"核"数。

3. 应用软件的问题

目前的绝大部分应用软件,特别是互联网应用软件都是面向多用户的多线程软件,这就需要计算机硬件系统在结构上能很好地支持线程化的软件,才能符合Amdahl定律的要求。

现在的数据库管理软件、数据库应用等服务器软件，一般要并行处理大量的、相对独立的事务。除了通用服务器软件外，还有大量的应用是线程化的。例如：

(1) 多线程的本地应用；

(2) 多进程应用；

(3) 多实例应用；

(4) Java 应用。

超线程(Hyper Threading)和 SMT 技术只能在一定程度上支持多线程或多实例应用，本质上还只是在一个执行核上运行。当线程个数较多时，就需要多核架构或并行处理机这样的处理系统了。

总之，受 CPU 主频、功耗、散热和超标量等技术复杂度的限制，以及多线程应用软件需求的驱动，微处理器架构发展到多核成为一种必然的趋势；另外，多核架构也是摩尔定律驱动的结果。在单核架构下，摩尔定律已经接近于失效，要想继续成立，多核架构是必然的选择。说到底，出现多核处理器最根本的原因是人们对计算能力永无止境的追求。

11.2.2　多核架构

顾名思义，多核技术是指在一枚处理器中集成两个或多个完整的计算内核，从而提高计算能力的技术。按计算内核的对等与否，多核架构又可以分为同构多核架构和异构多核架构两种。计算内核相同，地位对等的称为同构多核，反之称为异构多核。需要注意的是，多核架构与多处理器不同，多处理器指多个 CPU，每个 CPU 可以是单核或多核的。虽然同时使用多个 CPU，但是从管理的角度来看，它们的表现就像一台单机一样，这在前面的章节已经有所阐述。

1. 多核的组织架构

多核处理器的组织架构主要包括：片上核心处理器的个数、Cache 的级数、共享 Cache 的容量和内部互连结构等。图 11.1 给出了多核系统的 4 种典型的组织结构。

如图 11.1(a)所示是早期多核处理器的一种组织架构，现在在嵌入式芯片中仍能见到。在这种组织方式中，只有一级片内 Cache，每个核带有自己的专用 L1 Cache，分成指令 Cache 和数据 Cache。

如图 11.1(b)所示的是无片内共享 Cache 的组织结构。在这种结构里，片内有足够的可用面积容纳多个 L2 Cache。

如图 11.1(c)所示的架构采用了和图 11.1(b)类似的存储空间分配，不同的是该处理器架构拥有共享 L2 Cache。Intel 的 Core Duo 处理器就是这种结构。

最后，随着片上 CPU 内核总量的不断增加，出于性能上的考虑，分离出一个独立的三级 Cache，如图 11.1(d)所示；每个 CPU 计算内核除了拥有专用的一、二级 Cache 外，还共享 L3 Cache。

从上述几种结构来看，使用片内 Cache 是一种常见的技术和改善性能的方法。使用共享的片内 L2 Cache 相对于专用 Cache 而言有如下几个优点。

(1) 共享片内 L2 Cache 可以减少整个系统的不命中概率。也就是说，如果某个核上的

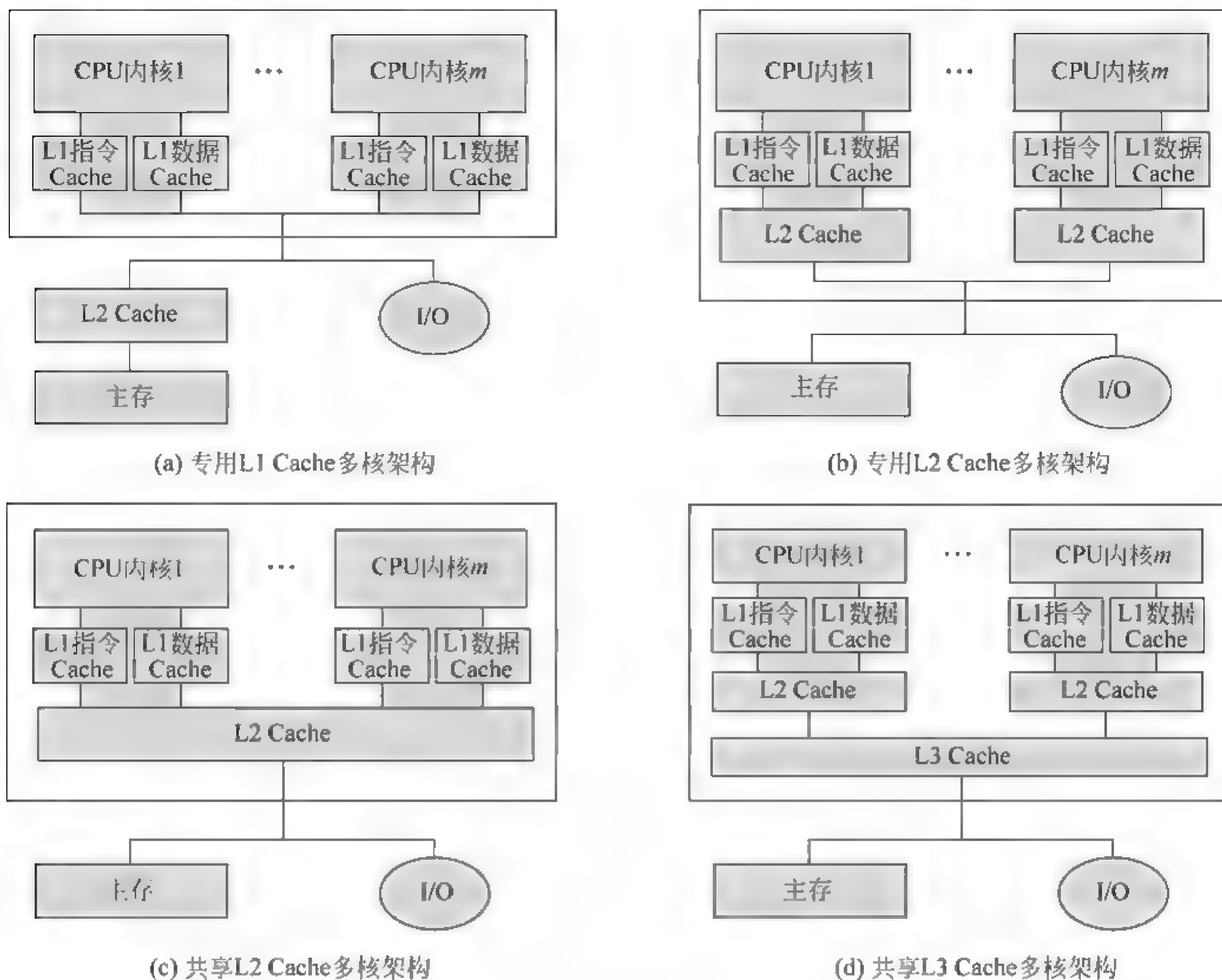


图 11.1　多核系统的组织架构

一个线程访问主存的某个位置，该位置对应的块会被装入共享 Cache；如果其他核上的线程也访问同一内存块，则数据已经在片内的共享 Cache 中了，这样就产生了 Cache 命中。

(2) 多个核所共享的数据在共享 Cache 级上不需要复制。

(3) 对于合适的块替换算法，分配给每个核的共享 Cache 量是动态的，这样局部线程能使用更多的 Cache 空间。

(4) 通过共享 Cache 能很容易地实现计算内核间的通信。

(5) 使用共享的 L2 Cache 将一致性问题限制在 L1 Cache 层次上，而且还具有性能上的优点，一方面不同核上的线程可以共享相同的数据，另一方面运行单个或少量高性能线程时，相应的 Cache 可用空间将更大。

使用片内专用 L2 Cache 的潜在优点是每个核能快速地访问其私有 L2 Cache 块，非常适合具有很强局部性的线程提高性能。

随着可用 Cache 的数量和计算内核数的增加，使用共享 L3 Cache，结合共享或专用的 L2 Cache，其效果会比简单地使用多个共享的 L2 Cache 要好，性能会得到进一步的提升。

多核系统另一个重要的组织架构设计问题是：每个计算内核采用超标量架构还是 SMT 架构。SMT 是一种在一个 CPU 的时钟周期内，能够执行来自多个线程指令的硬件多线程技术。因此，一个 4 核 SMT 系统，每个核支持 4 个并发线程，在应用级上等价于一个 16 核的系统。随着软件并行资源利用能力的提高，相比于超标量方法，SMT 方法更具有吸

引力。

2. 多核架构实例

多核 CPU 产品有很多，几乎所有的厂商都推出了自己的多核产品。本节介绍几个典型的多核架构实例：Intel x86 多核架构和面向嵌入式应用的 ARM 多核架构。

1) Intel x86 多核架构

从 2005 年开始，Intel 公司每两年推出一款新的微架构；工艺也从 65nm、45nm、32nm、22nm 逐步向 8nm 发展，性能上仍遵循摩尔定律。这里介绍两款分别代表中、低端应用的多核 CPU。

(1) Core Duo

2006 年推出的 Core Duo 是全球第一个低耗电的双核处理器(低于 25W)，它也是第一款苹果 Macintosh 电脑所使用的 Intel 处理器。Core Duo 实现了两个 x86 超标量处理器，共享二级 Cache。跟所有的多核系统一样，Core Duo 的每个核有自己的专用 L1 Cache：一个 32KB 的指令 Cache 和一个 32KB 的数据 Cache。

Core Duo 的每个核有一个独立的热控制部件。Core Duo 的热控制部件负责管理芯片的散热，在发热受限的条件下使得处理器的性能最高。另外，热管理可以通过冷却系统改进人机环境、降低风扇噪声。每个核可被定义为一个独立的热区，每个热区的最高温度存到专用寄存器中，由软件轮询这些寄存器来获得最高温度值。如果一个核的温度超过某个阈值，热控制部件降低该核的时钟频率，从而减少热量的产生。

Core Duo 组成中的另一个关键部件是高级可编程中断控制器(Advanced Programmable Interrupt Controller，APIC)。APIC 可以执行许多功能，包括支持处理器间中断，允许任一处理器中断另一处理器或一组处理器。一个核执行的线程可以产生一个中断请求，该中断请求首先由本地 APIC 接受，再传送给其他核的 APIC，然后中断对应的核。功率管理逻辑负责降低功耗，从而增加移动平台电池的寿命。在实际中，功率管理逻辑监测热量状况和 CPU 活动，适当地调整电压和功耗。它包含一个高级功率门控部件，可以进行超细粒度的逻辑控制，仅当需要的时候才启动相应的 CPU 逻辑子系统。

Core Duo 芯片包含一个共享的、2MB 的 L2 Cache。Cache 逻辑可以根据当前核的需求动态地分配 Cache 空间，因此一个核最多的时候可以获得全部的 L2 Cache 空间。L2 Cache 所包含的逻辑支持 MESI 协议，以维护其上 L1 Cache 之间的数据一致性。Core Duo 对 MESI 协议进行了扩展，也支持对称多处理器结构。当一个核请求的数据不在本地 CPU 内时，则通过外部总线上的代理访问其他 CPU。总线接口连接外部总线，即前端总线。前端总线连接主存、I/O 控制器和其他处理器芯片。

(2) Intel Core i7

Intel Core i7 是 Intel 于 2008 年 11 月推出的，实现了 4 个 x86 SMT 计算核，每个计算核带一个专用的 L2 Cache、一个共享的 L3 Cache。在 Core i7 中，每个核拥有自己的专用 L2 Cache，4 个核共享一个 8MB 的 L3 Cache。为了使 Cache 更加高效地工作，使用了预取机制。在这种机制中，硬件检测内存的访问模式，推测马上要用到的数据，并提前装入到 Cache 中。

Core i7 芯片支持两种片外通信方式：通过“DDR3 主存控制器”的通信和通过“高速路

径互连”的通信。Core i7 将 DDR3 主存控制器集成到了片内，去掉了前端总线。这个接口支持三个信道，每个信道为 8 字节宽，总宽度为 192 位，总数据传输率可达 32GB/s。高速路径互连(Quick Path Interconnect，QPI)是一个电气互连规范，基于一致性协议和点对点链路，用于 Intel 处理器和芯片组互连。通过其互连的处理器之间能高速通信，每秒可进行 6.4G 次传送。每次传送 16 位，达到 12.8GB/s；由于 QPI 链路是双向的，总带宽可达到 25.6GB/s。

2) ARM11 MPCore 架构

ARM11 MPCore 是基于 ARM11 处理器系列的多核产品，最多可配置 4 个处理器，每个处理器带有私有的 L1 指令 Cache 和 L1 数据 Cache。

ARM11 MPCore 处理器如图 11.2 所示，系统的主要部件如下。

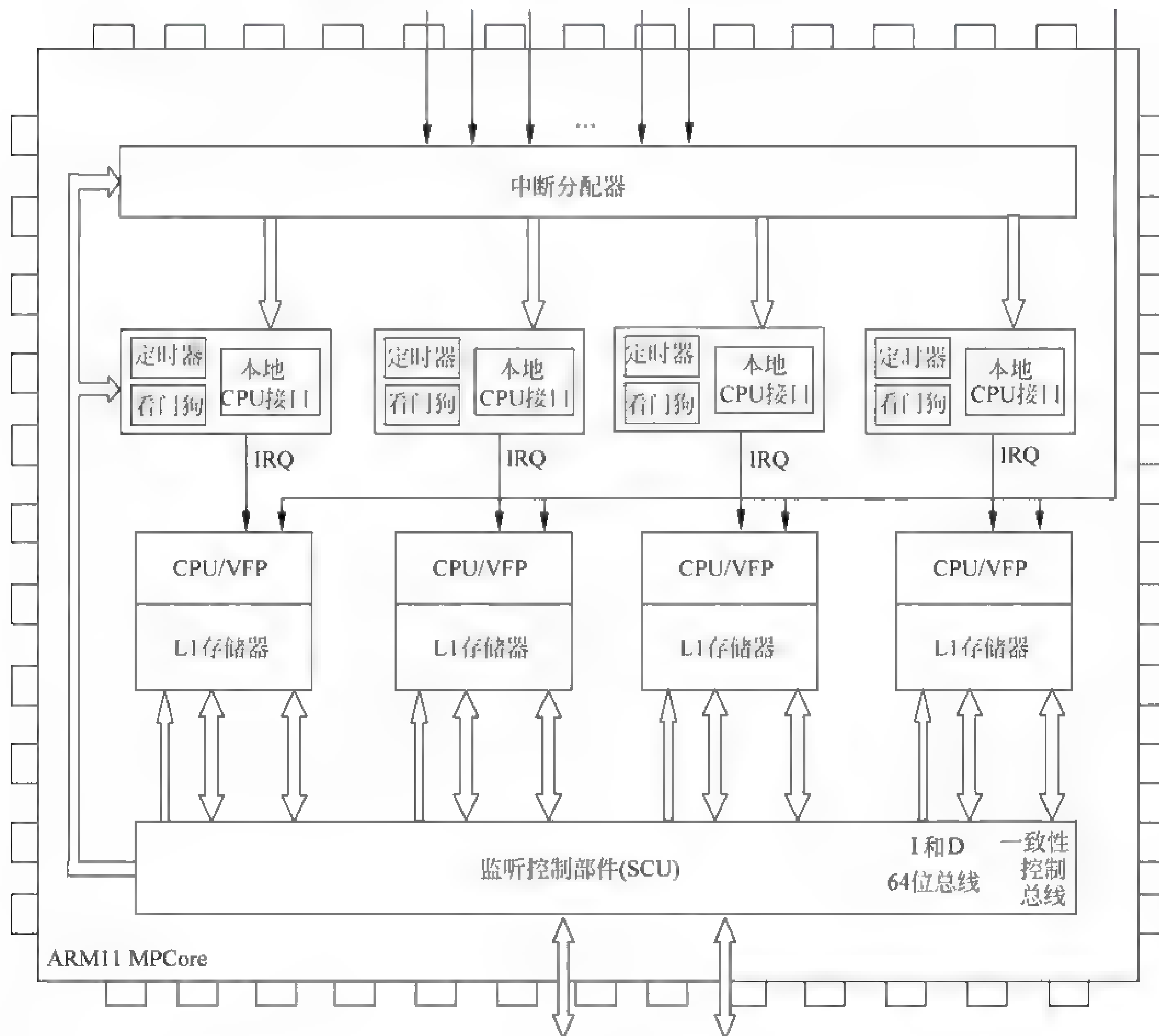


图 11.2　ARM11 MPCore 处理器

(1) 中断分配器：进行中断检测和中断优先级管理，按需将中断请求传递给一个或多个 CPU。提供一种处理器间的通信方法，使得一个 CPU 上线程可以驱动另一个 CPU 上的线程，支持单播、多播和广播通信方式。

(2) 定时器：每个 CPU 都有自己的、能产生中断的私有计时器。

(3) CPU 接口：处理中断确认、中断屏蔽和中断完成确认。

(4) CPU：单个的 ARM11 处理器。每个 CPU 被认为是 MP11 CPU 核。

(5) 向量浮点部件(VFP)：用硬件实现浮点运算的协处理器。

(6) L1 Cache：每个 CPU 有自己的专用 L1 数据 Cache 和 L1 指令 Cache。

(7) 监视控制部件：负责维护 L1 数据 Cache 之间的一致性。

11.2.3 基于多核的并行程序设计

毫无疑问，多核给我们提供了更经济的计算能力。但是，这种能力能否善加利用，还要取决于软件。如果不针对多核进行软件开发，不仅多核提供的强大计算能力得不到利用，相反还有可能不如单核 CPU 好用。因为采用多核的 CPU 其每个内核的主频比主流的单核 CPU 通常要低一些，如果你的程序只能发挥出一个内核效用，自然不如单核 CPU 好用。未来多核芯片将无处不在，针对多核的软件开发将是摆在软件产业界面前一个大的挑战。有专家甚至预言，针对多核和多线程的软件开发将是未来十年软件开发的主要挑战，即基于多核的并行程序设计。

多核处理器的基本目的是通过多个任务的并行执行提高应用程序的性能。这就需要将一个应用程序进行任务划分：尽量分解为多个相对独立的任务，每个任务实现为一个线程，从而将多个任务分布到多个计算核上执行，减少程序的执行时间。

1. 并行编程模型

目前几种最重要的并行编程模型是数据并行(Data Parallel)、消息传递(Message Passing)和共享变量(Shared Variable)。数据并行模型的编程级别比较高，编程相对简单，但它仅适用于数据并行问题；消息传递模型的编程级别相对较低，但消息传递编程模型可以有更广泛的应用范围；共享变量则采用多线程的方式，非常适合 SMP 共享内存多处理系统和多核处理器体系结构。

数据并行即将相同的操作同时作用于不同的数据，因此适合在 SIMD(Single Instruction Multiple Data)及 SPMD(Single Program，Multiple Data)的并行计算机上运行。数据并行编程模型是一种较高层次上的模型，它提供给编程者一个全局的地址空间，一般这种形式的语言本身就提供并行执行的语义，因此对于编程者来说，只需要简单地指明执行什么样的并行操作和并行操作的对象，就实现了数据并行的编程。数据并行的表达是相对简单和简洁的，它不需要编程者关心并行机是如何对该操作进行并行执行的。数据并行编程模型虽然可以解决一大类科学与工程计算问题，但是对于非数据并行类的问题，如果通过数据并行的方式来解决，一般难以取得较高的效率。

消息传递即各个并行执行的部分之间通过传递消息来交换信息、协调步伐、控制执行。消息传递一般是面向分布式内存的，但是它也可适用于共享内存的并行机。消息传递为编程者提供了更灵活的控制手段和表达并行的方法，一些用数据并行方法很难表达的并行算法，都可以用消息传递模型来实现，灵活性和控制手段的多样化，是消息传递并行程序能提供高的执行效率的重要原因。消息传递模型一方面为编程者提供了灵活性；另一方面，它

也将各个并行执行部分之间复杂的信息交换和协调、控制的任务交给了编程者,这在一定程度上增加了编程者的负担,这也是消息传递编程模型编程级别低的主要原因。虽然如此,消息传递的基本通信模式是简单和清楚的,学习和掌握这些部分并不困难,因此目前大量的并行程序设计仍然是消息传递并行编程模式。

共享变量则是采用程序中的共享变量来进行信息交换、协调同步以及控制执行的。共享变量的方式非常适合于多核系统下的应用编程,著名的OpenMP多线程并行编程语言就是采用的这种方式。

2. 并行语言

并行程序是通过并行语言来表达的,并行语言的产生主要有以下3种方式。

(1) 设计全新的并行语言;

(2) 扩展原来的串行语言的语法成分使它支持并行特征;

(3) 不改变串行语言仅为串行语言提供可调用的并行库。

设计一种全新的并行语言的优点是可以完全摆脱串行语言的束缚,从语言成分上直接支持并行,这样就可以使并行程序的书写更方便,更自然,相应的并行程序也更容易在并行机上实现。但是,由于并行计算至今还没有像串行计算那样统一的冯·诺依曼模型可供遵循,因此并行机、并行模型、并行算法和并行语言的设计和开发千差万别,没有一个统一的标准,虽然有多种多样全新的并行语言出现,但至今还没有任何一种新出现的并行语言,成为普遍接受的标准,设计全新的并行语言,实现起来难度和工作量都很大,但各种各样并行语言的出现、实践和研究无疑都为并行语言和并行计算的发展做出了贡献。

一种重要的对串行语言的扩充方式就是标注,即将对串行语言的并行扩充作为原来串行语言的注释,对于这样的并行程序,若用原来的串行编译器来编译,标注的并行扩充部分将不起作用,仍将该程序作为一般的串行程序处理,若使用扩充后的并行编译器来编译,则该并行编译器就会根据标注的要求,将原来串行执行的部分转化为并行执行。对串行语言的并行扩充,相对于设计全新的并行语言,显然难度有所降低,但需要重新开发编译器,使它能够支持扩充的并行部分。一般地,这种新的编译器往往和运行时支持的并行库相结合。仅提供并行库,是一种对原来的串行程序设计改动最小的并行化方法。这样,原来的串行编译器也能够使用,不需要任何修改,编程者只需要在原来的串行程序中加入对并行库的调用,就可以实现并行程序设计。对于这三种并行语言的实现方法目前最常使用的是第二种和第三种方法,特别是第三种方法。

3. 并行算法

并行算法是给定并行模型的一种具体、明确的解决方法和步骤。按照不同的划分方法,并行算法有多种不同的分类。

根据运算的基本对象的不同,可以将并行算法分为数值并行算法(数值计算)和非数值并行算法(符号计算)。当然,这两种算法也不是截然分开的。划分为什么类型的算法主要取决于主要的计算量和宏观的计算方法。

根据进程之间的依赖关系可以分为同步并行算法(步调一致)、异步并行算法(步调、进展互不相同)和纯并行算法(各部分之间没有关系)。对于同步并行算法,任务的各个部分是

同步向前推进的，有一个全局的时钟(不一定是物理的)来控制各部分的步伐；而对于异步并行算法，各部分的步伐是互不相同的，它们根据计算过程的不同阶段决定等待、继续或终止；纯并行算法是最理想的情况，各部分之间可以尽可能快地向前推进，不需要任何同步或等待，但是一般这样的问题是少见的。

根据并行计算任务的大小，还可以分为粗粒度并行算法(一个并行任务包含较长的程序段和较大的计算量)、细粒度并行算法(一个并行任务包含较短的程序段和较小的计算量)以及介于二者之间的中粒度并行算法。一般而言，并行的粒度越小，就有可能开发更多的并行性，提高并行度，这是有利的方面，但是另一个不利的方面就是并行的粒度越小，通信次数和通信量就相对增多，这样就增加了额外的开销，因此合适的并行粒度需要根据计算量、通信量、计算速度、通信速度进行综合平衡，这样才能够取得高效率。

对于相同的并行计算模型，可以有多种不同的并行算法来描述和刻画。由于并行算法设计的不同，可能对程序的执行效率有很大的影响，不同的算法可以有几倍、几十倍甚至上百倍的性能差异是完全正常的。并行算法基本上是随着并行机的发展而发展的。从本质上说，不同的并行算法是根据问题类别的不同和并行机体系结构的特点产生出来的，一个好的并行算法要既能很好地匹配并行计算机硬件体系结构的特点，又能反映问题内在并行性。SIMD 结构计算机一般适合同步并行算法，而 MIMD 并行计算机则适合异步并行算法。

在并行计算中，由于并行算法可以对性能产生重大的影响，因此受到广泛的重视，并行算法也成为一个专门的十分活跃的研究领域。因此在并行程序设计之前，必须首先考虑好并行算法。该算法要能够将并行机和实际的问题很好地结合起来，既能够充分利用并行机体系结构的特点，又能够揭示问题内在的并行性。

11.2.4　多核编程实例

目前，程序开发人员开发实际的并行程序主要方法是串行语言加并行库的扩展，也就是增加一个库来帮助进行消息传递和并行，但其程序开发效率很低，难度也比较大。其中比较典型的方法有两种：共享存储和消息传递。共享存储的方法主要是采用多线程的方式，其主要程序开发环境就是已经成为事实工业标准的 OpenMP，目前主要是商业编译器提供对该语言的支持。而消息传递开发则包括 MPI 和 PVM 等开源开发环境，可以免费下载。其中，MPI 最常用最流行的两个实现是 MPICH 和 LAM/MPI。此外，由于现有机器体系结构层次非常复杂，还可以把上面几类并行和向量并行混合使用，充分挖掘机器的性能潜力。通常称之为混合并行。

OpenMP(Open Multi Processing)是一套支持跨平台共享内存方式的多线程并发的编程 API，使用 C，C++ 和 FORTRAN 语言，可以在大多数的处理器体系和操作系统中运行，包括 Solaris，AIX，HP UX，GNU/Linux，Mac OS X 和 Microsoft Windows 等；还包括一套编译器指令、库和一些能够影响运行行为的环境变量。OpenMP 采用可移植的、可扩展的模型，为程序员提供了一个简单而灵活的开发平台，包括从标准桌面 PC 到超级计算机的并行应用程序接口。OpenMP 提供了对并行算法的高层的抽象描述，程序员通过在源代码中加入专用的 pragma 来指明自己的意图，由此编译器可以自动将程序进行并行化，并在必要之处加入同步互斥以及通信。

习　　题

1. 概念题

【题 11.1】 解释下列名词

CMP　　SMT　　MPI　　OpenMP

2. 问答题

【题 11.2】 并行编程模型有哪几种？各自的特点是什么？
【题 11.3】 简述多核架构与多处理器有什么不同。
【题 11.4】 多核时代的主要驱动力主要有哪些？
【题 11.5】 画出专用 L1 Cache 多核架构图，并叙述其特点。
【题 11.6】 画出专用 L2 Cache 多核架构图，并叙述其特点。
【题 11.7】 画出共享 L2 Cache 多核架构图，并叙述其特点。
【题 11.8】 画出共享 L3 Cache 多核架构图，并叙述其特点。

题　　解

1. 概念题

【题 11.1】 解释下列名词

CMP -- 多核处理器又称芯片多处理器(Chip Multiprocessor，CMP)，是指在单个芯片内集成两个或多个处理器。

SMT -- 超标量技术以及同时多线程(Simultaneous Multithreading，SMT)。

MP——消息传递并行程序设计(Message Passing Interface，MPI)。

OpenMP——多线程并行编程语言(Open Multi-Processing，OpenMP)。

2. 问答题

【题 11.2】 答：目前有 3 种并行编程模型：数据并行(Data Parallel)模型、消息传递(Message Passing)模型和共享变量(Shared Variable)模型。数据并行模型的编程级别比较高，编程相对简单，但它仅适用于数据并行问题；消息传递模型的编程级别相对较低，但消息传递编程模型可以有更广泛的应用范围；共享变量则采用多线程的方式，非常适合 SMP 共享内存多处理系统和多核处理器体系结构。

【题 11.3】 答：多核是指在一片处理器芯片中集成多个完整的计算内核。

而多处理器则是指多个 CPU，每个 CPU 可以是单核或多核的。虽然同时使用多个 CPU，但是从管理的角度来看，它们的表现就像一台单机一样。

【题 11.4】 答：主要有功耗与散热、并行度和应用软件等。受 CPU 主频、功耗、散热和

超标量等技术复杂度的限制，以及多线程应用软件需求的驱动，微处理器架构发展到多核成为一种必然；另外，多核架构也是摩尔定律驱动的结果。在单核架构下，摩尔定律已经接近于失效了，要继续成立，多核也是必然的选择。

【题 11.5】 答：专用 L1 Cache 多核架构图如图 11.3 所示。

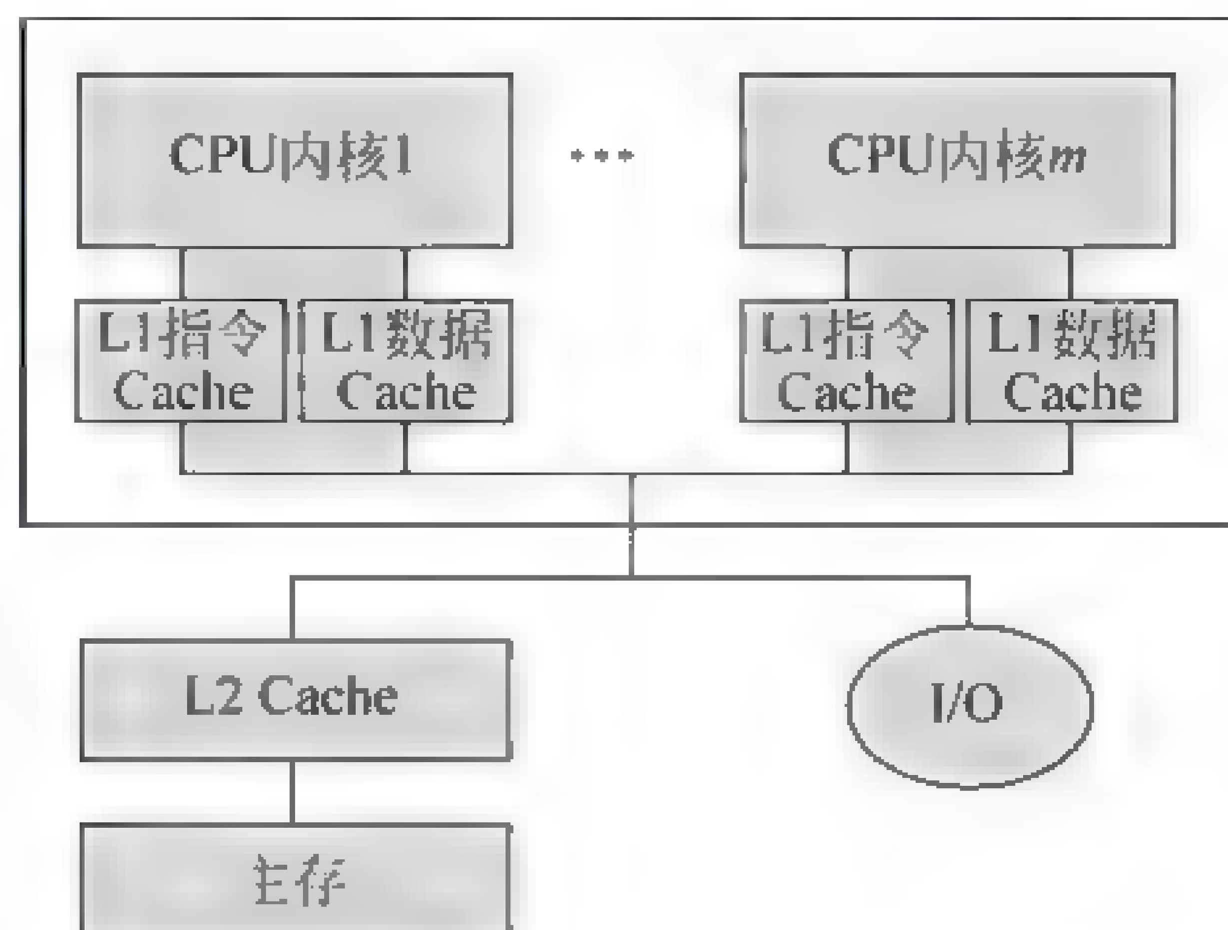


图 11.3　专用 L1 Cache 多核架构图

特点：(1) 每个 CPU 核都有专用的 L1 指令 Cache 和 L1 数据 Cache，访问带宽比共享 L1 Cache 提高一倍。

(2) 片内无 L2 Cache。

(3) 共享片外 L2 Cache，不存在 L2 Cache 一致性的问题。

【题 11.6】 答：专用 L2 Cache 多核架构图如图 11.4 所示。

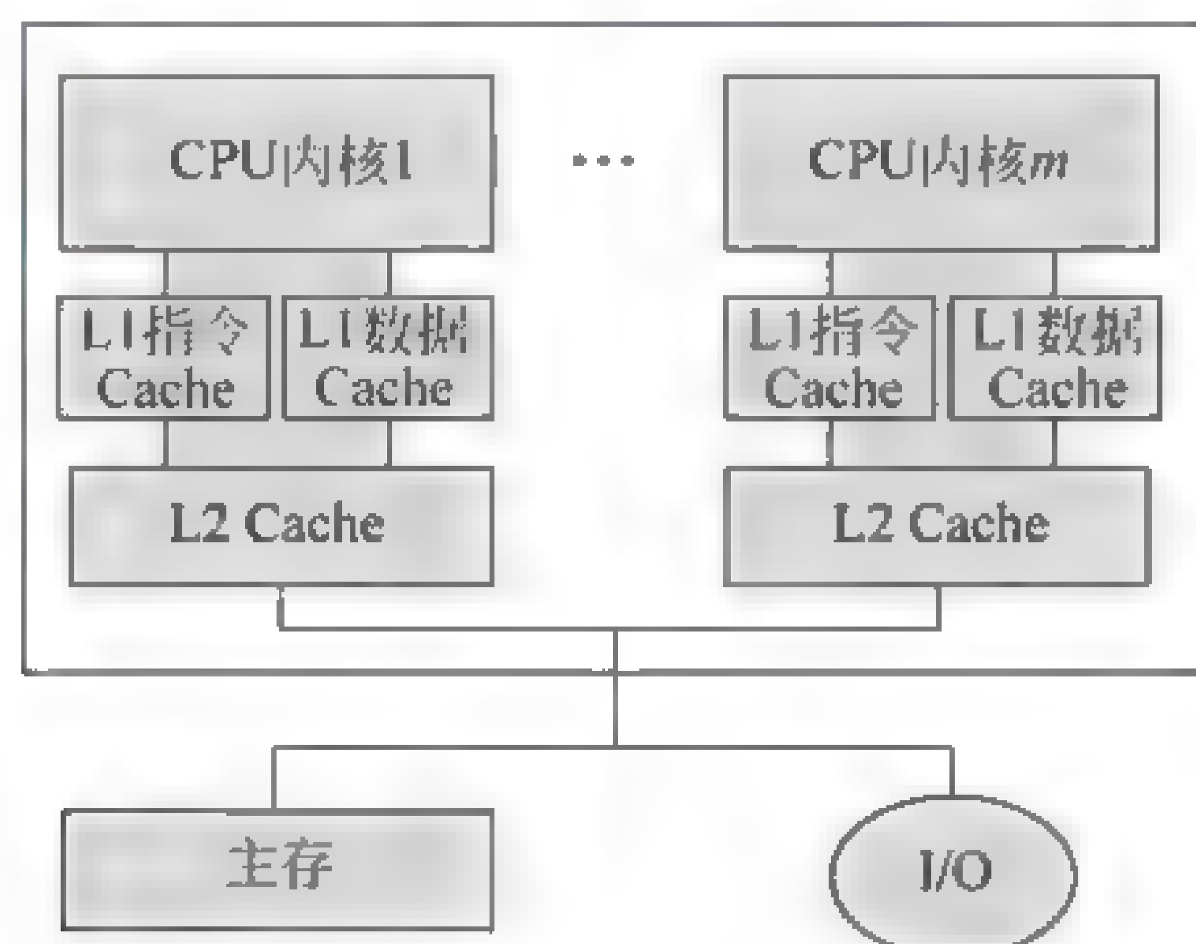


图 11.4　专用 L2 Cache 多核架构图

特点：(1) 每个 CPU 核都有专用的 L1 指令 Cache 和 L1 数据 Cache，访问带宽比共享 L1 Cache 提高一倍。

(2) 每个 CPU 核都有一个片内 L2 Cache，增加了每个核的 Cache 容量，但也带来了实现多 L2 Cache 一致性的问题。

【题 11.7】 答：共享 L2 Cache 多核架构图如图 11.5 所示。

特点：(1) 每个 CPU 核都有专用的 L1 指令 Cache 和 L1 数据 Cache，访问带宽比共享 L1 Cache 提高一倍。

(2) 所有 CPU 核共享一个片内 L2 Cache，不存在 L2 Cache 一致性的问题。

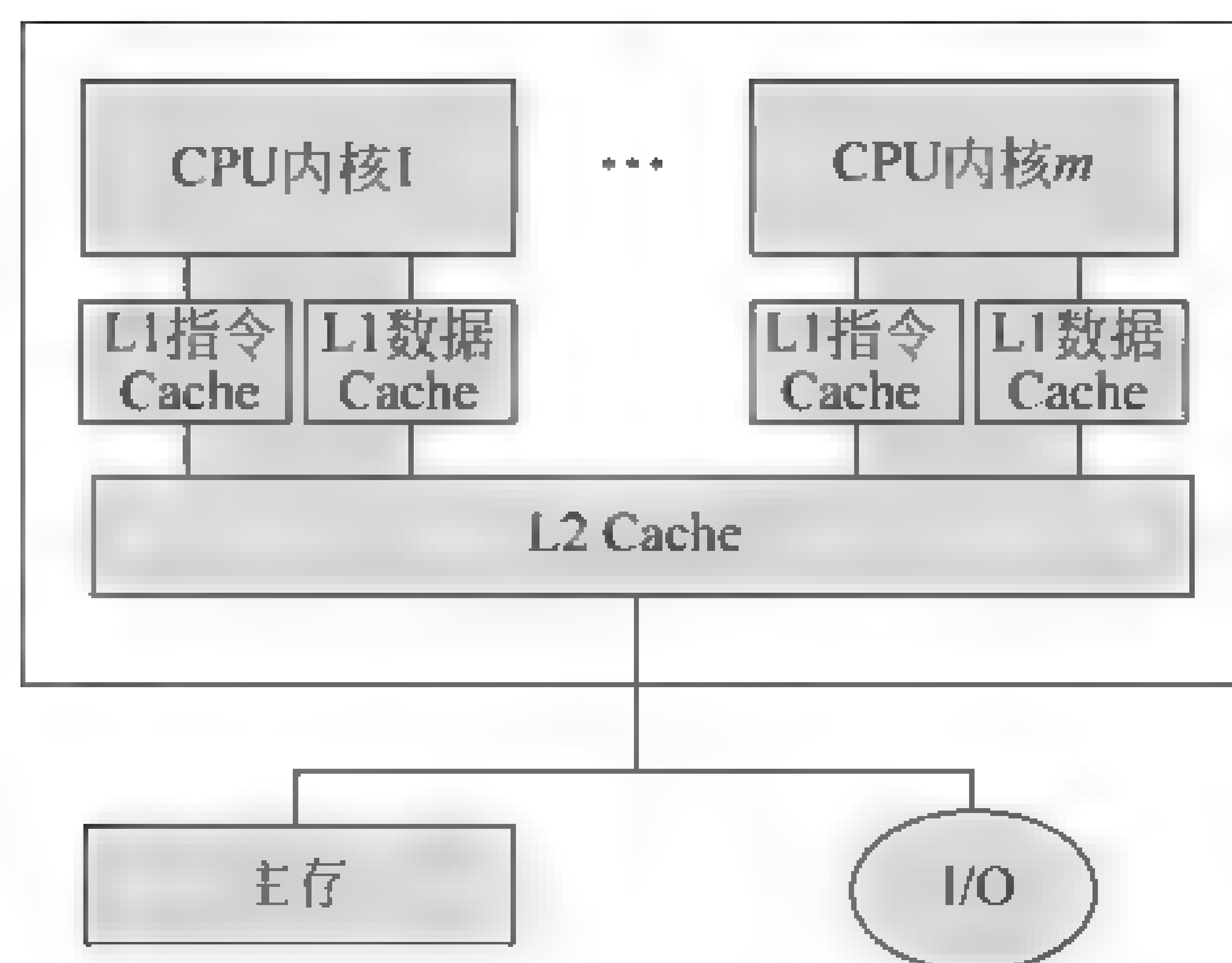


图 11.5 共享 L2 Cache 多核架构图

【题 11.8】 答：共享 L3 Cache 多核架构图如图 11.6 所示。

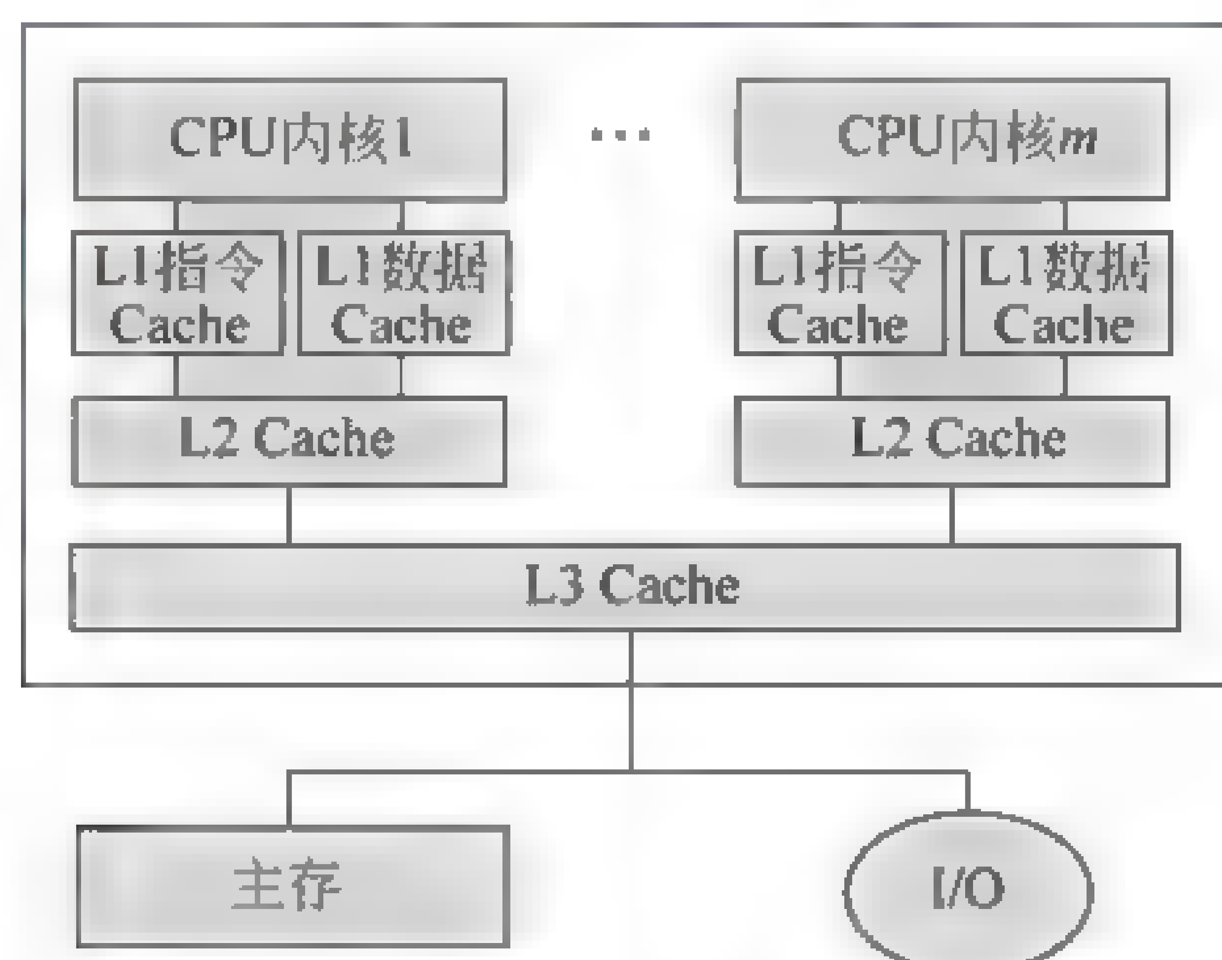


图 11.6 共享 L3 Cache 多核架构图

特点：(1) 每个 CPU 核都有专用的 L1 指令 Cache 和 L1 数据 Cache，访问带宽比共享 L1 Cache 提高一倍。

(2) 每个 CPU 核都有一个片内 L2 Cache，增加了每个核的 Cache 容量，但也带来了实现多 L2 Cache 一致性的问题。

(3) 所有 CPU 核共享一个片内 L3 Cache，进一步增加了 Cache 的容量，而且也不存在 L2 Cache 一致性的问题。

第12章　机群系统

12.1　基本要求与难点

12.1.1　基本要求

(1) 掌握有关机群系统的基本概念。
(2) 掌握机群的基本结构，包括硬件组成和软件组成。
(3) 掌握机群的特点和分类。
(4) 了解典型的机群系统，包括：Berkeley NOW，Beowulf，LAMP，IBM SP2。

12.1.2　难点

机群的特点和分类。

12.2　知识要点

机群(Cluster)系统起源于20世纪90年代中期，它是由多台同构或异构的独立计算机通过高性能网络连接在一起而构成的高性能并行计算机系统。构成机群的计算机都拥有自己的存储器、I/O设备和操作系统，它们在机群操作系统的控制下协同完成特定的并行计算任务。对用户和应用来说，机群就是一个单一的系统，可以提供低价高效的高性能环境和快速可靠的服务。

目前流行的高性能并行计算机系统结构通常可以分成5类：并行向量处理机(PVP)、对称多处理机(SMP)、大规模并行处理机(MPP)、分布共享存储(DSM)多处理机和机群。机群结构凭借低廉的价格、极强的灵活性和可扩放性，成为近年来发展势头最为强劲的一种结构。

尽管直到1997年6月全球高性能计算机500强(Top500)才首次有机群结构的计算机进入该排名，但此后入选的机群系统的数量逐年稳步增加——2003年11月，这一数字已达到208台，机群首次成为Top500排名中比例最高的结构；截至2008年6月，已经连续10期位居榜首。机群已成为当今构建高性能计算机系统时最常被采用的结构。

12.2.1　机群的基本结构

1. 机群的硬件组成

机群是一种价格低廉、易于构建、可扩放性极强的并行计算机系统,它由多台同构或异构的独立计算机通过高性能网络或局域网互连在一起,协同完成特定的并行计算任务。从用户的角度来看,机群就是一个单一、集中的计算资源。

构成机群的每台独立计算机都是机群的一个结点。每个结点都是一个完整的计算机系统,拥有本地磁盘和操作系统,可以作为一个单独的计算资源供用户使用。除了PC外,机群的结点还可以是工作站,甚至是规模较大的对称多处理机。

按照机群系统中各结点的功能不同,可以将它们分为3类,即:①计算结点,用于完成计算任务。②管理/登录结点,它们是外部设备和机群系统之间连接的桥梁,任何用户和系统的管理员都只能通过此类结点才能登录到系统中。此外,管理/登录结点还应具有管理和作业提交等功能。③I/O结点,作为NFS文件系统的主结点,I/O结点一般由存储设备、网络文件系统(NFS)等组成,外挂磁盘阵列或者连接其他存储设备,负责文件的I/O操作,其他结点访问存储设备的请求都要通过I/O结点完成。

这三类结点所需的具体硬件配置也不相同。计算结点需要提供很强的计算能力,对于某些应用而言特别需要强大的浮点计算能力。此外,计算结点还应提供适量的内存,使运算时数据能完全驻留在物理内存中,并能够支持高速、低延迟的通信网络。而系统对管理/登录结点要求不高,只要采用相对经济的配置就可以了。

机群的各个结点一般通过商品化网络连接在一起,如以太网、Myrinet、Infiniband、Quadrics等,部分商用机群也采用专用网络连接,如SP Switch、NUMAAlink、Crossbar、Cray Interconnect等。网络接口与结点的I/O总线以松耦合的方式相连。

无论是计算机还是互连网络,可供设计者选择的产品都非常多,而且不同厂家的产品在功能、性能以及价格上也都有所差别,如何选择合适的产品,主要取决于具体的用户对机群的具体要求。

2. 机群的软件

软件也是机群系统的重要组成部分。由于机群系统结构松散、结点独立性强、网络连接复杂,造成机群系统管理不便、难以使用。为了解决这一问题,国际上流行的方式是在各结点的操作系统之上再建立一层操作系统来管理整个机群,这就是机群操作系统。

除了提供硬件管理、资源共享以及网络通信等功能外,机群操作系统还必须完成的另外一项重要功能是实现**单一系统映像**(Single System Image,SSI),这是机群的一个重要特征。正是通过SSI才使得机群在使用、控制、管理和维护上更像一个单独的计算资源。

SSI共有四重含义。首先是"单一系统",尽管系统中有多个处理器,用户仍然把整个机群视为一个单一的计算系统来使用。其次是"单一控制",在逻辑上,最终用户或系统用户使用的服务都来自机群中唯一的一个位置。例如,用户将批处理作业提交到一个唯一的作业集中,而系统管理员则通过一个唯一的控制点对整个机群的所有软、硬件进行管理和配置。

三是"对称性",用户可以从机群的任何一个结点上获得服务,也就是说,对于所有结点和所有用户,除了那些具有特定访问权限的服务与功能外,所有其他服务与功能都是对称的,可以通过任何一个结点提供给用户。最后则是"位置透明",用户不必了解真正执行服务的物理设备的具体位置。

一般来说,机群系统中的 SSI 至少应该提供以下 3 种服务。

(1) 单一登录(Single Sign On),即用户可以通过机群中的任何一个结点登录,而且在整个作业执行过程中只需登录一次,不必因作业被分派到其他结点上执行而重新登录。

(2) 单一文件系统(Single File System),这有两方面含义。首先,在机群系统中,有一些对整个机群所有结点都相同的软件,没有必要在每一个结点上重复安装。其次,尽管执行并行作业时要求每个结点都可以访问到这些软件,但它们在整个机群系统中应该只有一个备份。

(3) 单一作业管理系统(Single Job Management System)。用户可以透明地从任一结点提交作业,作业可以以批处理、交互或并行的方式被调度执行。PBS、LSF、Condor 和 JOSS 都是目前比较具有代表性的作业管理系统。

此外,并行编程模型以及相关的并行编程环境也是机群系统中不可缺少的软件。目前比较流行的并行编程工具包括 MPI、PVM、OpenMP、HPF 等。MPI(Message Passing Interface)是目前最重要的一个基于消息传递的并行编程工具,它具有可移植性好、功能强大、效率高等许多优点,而且有许多不同的免费、高效、实用的实现版本,几乎所有的并行计算机厂商都提供对它的支持,使它成为并行编程的事实标准。PVM(Parallel Virtual Machine)也是一种常用的基于消息传递的并行编程环境,它把工作站网络构建成一个虚拟的并行机系统,为并行应用程序提供了运行平台。HPF(High Performance FORTRAN)是一个支持数据并行的并行语言标准。OpenMP(Open Multi Processing)是一个共享存储并行系统上的应用编程接口,它规范了一系列的编译制导、运行库例程和环境变量,并为 C/C++ 和 FORTRAN 等高级语言提供了应用编程接口,已经应用于 UNIX、Windows 等多种平台。

图 12.1 列出了机群系统的软件框架。机群操作系统、SSI 以及其他一些机群正常工作所必需的软件一同构成了机群中间件。在它之上是并行编程环境,用户可以通过并行编程环境完成并行应用程序的开发。当然,串行应用也可以通过机群中间件被调度到某个结点上执行。

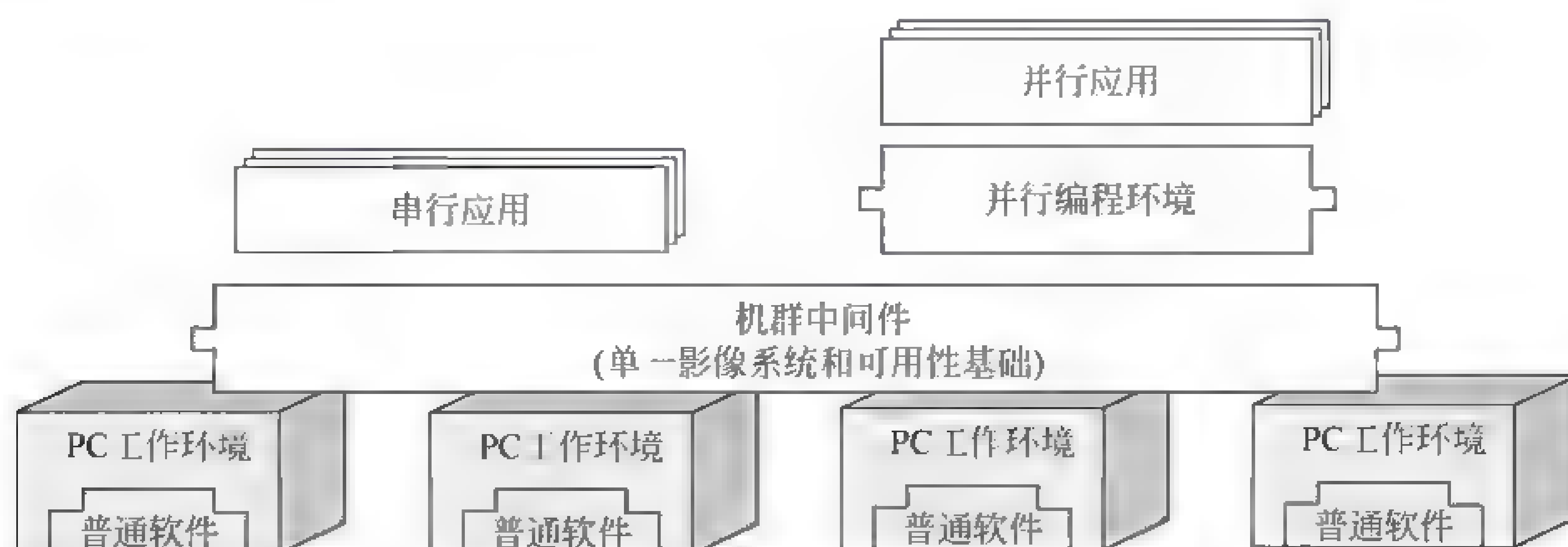


图 12.1 机群系统的软件框架

12.2.2　机群的特点

与MPP、PVP、SMP、DSM等传统并行计算机系统相比，机群系统具有许多优点。

(1) 系统开发周期短。由于机群系统大多采用商品化的PC、工作站作为结点，并通过商用网络连接在一起，系统开发的重点在于通信子系统和并行编程环境上，这大大节省了研制时间。

(2) 可靠性高。机群中的每个结点都是独立的PC或工作站，某个结点的失效并不会影响其他结点的正常工作，而且它的任务还可以迁移到其他结点上继续完成，从而有效地避免由于单结点失效引起的系统可靠性问题。

(3) 可扩放性强。机群的计算能力随着结点数量的增加而增大。这一方面得益于机群结构的灵活性，由于结点间以松耦合方式连接，机群的结点数量可以增加到成百上千；另一方面则是由于机群系统的硬件容易扩充和替换，可以灵活配置。

(4) 性能价格比高。由于生产批量小，传统并行计算机系统的价格均比较昂贵，往往要几百万到上千万美元。而机群的结点和网络都是商品化的计算机产品，能够大批量生产，成本相对较低，因而机群系统的性能价格比更好。与相同性能的传统并行计算机系统相比，机群的价格要低1～2个数量级。

(5) 用户编程方便。机群系统中，程序的并行化只是在原有的C、C++或FORTRAN串行程序中插入相应的通信原语，对原有串行程序的改动有限。用户仍然使用熟悉的编程环境，无须适应新的环境。

但是机群也有不足之处。由于机群由多台完整的计算机组成，它的维护相当于要同时去管理多个计算机系统，因此维护工作量较大，维护费用也较高。SMP则相对较好，因为管理员只要维护一个计算机系统即可。正因为如此，现在很多机群采用SMP作为结点，这样可以减少结点数量，达到减少维护工作量和开支的目的。

12.2.3　机群的分类

按照不同的标准，机群的分类方法有很多。例如，根据组成机群的各个结点和网络是否相同，机群可以分为同构与异构两类；根据结点是PC还是工作站，机群可以进一步分为PC机群与工作站机群。不过最常用的分类方法还是以机群系统的使用目的为依据，将其分为高可用性机群、负载均衡机群以及高性能机群三类。

(1) 高可用性机群。这类机群的主要目的是在系统中某些结点出现故障的情况下，仍能继续对外提供服务。它采用冗余机制，当系统中某个结点由于软、硬件故障而失效时，该结点上的任务将在最短的时间内被迁移到机群内另一个具有相同功能与结构的结点上继续执行。这样，对于用户而言，系统可以一直为其提供服务。这类机群适用于Web服务器、医学监测仪、银行POS系统等要求持续提供服务的应用。

(2) 负载均衡机群。这类机群的主要目的是提供与结点个数成正比的负载能力，这就要求机群能够根据系统中各个结点的负载情况实时地进行任务分配。为此，它专门设置了一个重要的监控结点，负责监控其余每个工作结点的负载和状态，并根据监控结果将任务分

派到不同的结点上。这种机群很适合大规模网络应用(如 Web 服务器或 FTP 服务器)、大工作量的串行或批处理作业(如数据分析)。

负载均衡机群往往也具有一定的高可用性特点,但二者的工作原理不同,因而适用于不同类型的服务。通常,负载均衡机群适用于提供静态数据的服务,如 HTTP 服务;而高可用性机群既适用于提供静态数据的服务,又适用于提供动态数据的服务,如数据库等。之所以高可用性集群能适用于提供动态数据的服务,是由于它的结点共享同一存储介质。也就是说,在高可用性机群内,每种服务的用户数据只有一份,存放专门的存储结点上,在任一时刻只有一个结点能读写这份数据。

(3) 高性能计算机群。这类机群的主要目的是降低高性能计算的成本。它通过高速的商用互连网络,将数十个乃至上千台 PC 或工作站连接在一起,可以提供接近甚至超过传统并行计算机系统的计算能力,但其价格却仅是具有相同计算能力的传统并行计算机系统的几十分之一。这样,通过利用若干台 PC 就可以完成通常只有超级计算机才能完成的计算任务。这类机群适用于计算量巨大的并行应用,如石油矿藏定位、气象变化模拟、基因序列分析等。当然,为了稳定地提供高性能计算服务,它也必须满足一定的可用性要求。

还有一种比较常用的分类方法是按照构建方式将机群分为专用机群和企业机群两类。**专用机群**是为代替传统的大中型计算机或巨型计算机而设计的,装置比较紧凑,一般都装在比较小的机架内,放在机房中使用,因此它的吞吐率较高,响应时间也较短。专用机群的结点往往是同构的,一般采用集中控制,由一个(或一组)管理员统一管理,而且用户一般需要通过一台终端机来访问它,这样做的好处是其内部对外界完全屏蔽。而**企业机群**则正好相反,它是为了充分利用各个结点的空闲资源而设计的,因此其各个结点分散安放,并不需要安装在同一个房间,甚至不需要安排在同一幢楼中。各结点一般通过标准的 LAN 或 WAN 互连,通信开销较大、延迟较长。企业机群的各个结点一般是异构的,并由不同的个人拥有,这样机群管理者只能对各个结点进行有限的管理,结点拥有者可以随意地进行关机、重新配置或者升级,而且对一个结点而言,它的拥有者的任务应该具有最高优先级,高于企业的其他用户。显然,企业机群的内部通信是对外暴露的,存在一定的安全隐患,需要在通信子系统中采用专门措施来避免。

习　　题

1. 概念题

【题 12.1】 解释下列名词

机群	单一系统映像	高可用性机群
负载均衡机群	高性能机群	Beowulf 机群

2. 问答题

【题 12.2】 机群系统有哪些特点?

【题 12.3】 说明 IBM SP2 机群在体系结构上的特点。

【题12.4】 机群系统的SSI提供了哪些服务?

【题12.5】 机群系统中的结点可以按照功能分为哪些类型?它们在配置上有何不同?

【题12.6】 根据机群系统的使用目的可以将机群系统分为哪三类?它们分别有什么特点?

【题12.7】 试描述专用机群和企业机群的区别。

题　　解

1. 概念题

【题12.1】 解释下列名词

机群——机群是由多台同构或异构的独立计算机通过高性能网络连接在一起而构成的高性能并行计算机系统。构成机群的计算机都拥有自己的存储器、I/O设备和操作系统,它们在机群操作系统的控制下协同完成特定的并行计算任务。对用户和应用来说,机群就是一个单一的系统,可以提供低价高效的高性能环境和快速可靠的服务。

单一系统映像——单一系统映像具有四重含义。首先是"单一系统",尽管系统中有多个处理器,用户仍然把整个机群视为一个单一的计算系统来使用。其次是"单一控制",在逻辑上,最终用户或系统用户使用的服务都来自机群中唯一的一个位置。三是"对称性",用户可以从机群的任何一个结点上获得服务。最后则是"位置透明",用户不必了解真正执行服务的物理设备的具体位置。

高可用性机群——在系统中某些结点出现故障的情况下,仍能继续对外提供服务。它采用冗余机制,当系统中某个结点由于软、硬件故障而失效时,该结点上的任务将在最短的时间内被迁移到机群内另一个具有相同功能与结构的结点上继续执行。

负载均衡机群——提供与结点个数成正比的负载能力,这就要求机群能够根据系统中各个结点的负载情况实时地进行任务分配。为此,它专门设置了一个重要的监控结点,负责监控其余每个工作结点的负载和状态,并根据监控结果将任务分派到不同的结点上。

高性能机群——这类机群的主要目的是降低高性能计算的成本。它通过高速的商用互连网络,将数十个乃至上千台PC或工作站连接在一起,可以提供接近甚至超过传统并行计算机系统的计算能力,但其价格却仅是具有相同计算能力的传统并行计算机系统的几十分之一。

Beowulf机群——Beowulf机群定义了这样一种系统:使用普通的硬件加上Linux操作系统、再加上GNU开发环境以及PVM/MPI共享库所构建的机群。它一方面集中了那些相对较小的机器的计算能力,能够以很高的性能价格比提供与大型计算机相当的性能,另一方面也保证了软件环境的稳定性。Beowulf并不是一套具体的软件包或是一种新的网络拓扑结构,它只是一种思想,即:在达到既定目标的前提下,把注意力集中在获取更高的性能价格比上。

2. 问答题

【题12.2】 答：机群系统具有许多优点。

（1）系统开发周期短。由于机群系统大多采用商品化的PC、工作站作为结点，并通过商用网络连接在一起，系统开发的重点在于通信子系统和并行编程环境上，这大大节省了研制时间。

（2）可靠性高。机群中的每个结点都是独立的PC或工作站，某个结点的失效并不会影响其他结点的正常工作，而且它的任务还可以迁移到其他结点上继续完成，从而有效地避免由于单结点失效引起的系统可靠性问题。

（3）可扩放性强。机群的计算能力随着结点数量的增加而增大。这一方面得益于机群结构的灵活性，由于结点间以松耦合方式连接，机群的结点数量可以增加到成百上千；另一方面则是由于机群系统的硬件容易扩充和替换，可以灵活配置。

（4）性能价格比高。由于生产批量小，传统并行计算机系统的价格均比较昂贵，往往要几百万到上千万美元。而机群的结点和网络都是商品化的计算机产品，能够大批量生产，成本相对较低，因而机群系统的性能价格比更好。与相同性能的传统并行计算机系统相比，机群的价格要低1～2个数量级。

（5）用户编程方便。机群系统中，程序的并行化只是在原有的C、C++或FORTRAN串行程序中插入相应的通信原语，对原有串行程序的改动有限。用户仍然使用熟悉的编程环境，无须适应新的环境。

【题12.3】 答：IBM SP2是机群中的代表性产品，它既可用于科学计算，也可供商业应用。SP2机群是异步的MIMD，具有分布式存储器系统结构。它的每个结点都是一台RS/6000工作站，带有自己的存储器和本地磁盘。结点中采用的处理器是一种6流出的超标量处理机。每个结点配有一套完整的AIX操作系统（IBM的UNIX），结点间的互连网络接口是松耦合的，通过结点本身的I/O微通道（MCC）接到网络上，而不是通过本身的存储器总线。SP2中设置了一个专门的系统控制台用以管理整个系统。通过该系统控制台，系统管理人员可以从单一地点对整个系统进行管理。

【题12.4】 答：机群系统中的SSI至少应该提供以下三种服务。

（1）单一登录。即用户可以通过机群中的任何一个结点登录，而且在整个作业执行过程中只需登录一次，不必因作业被分派到其他结点上执行而重新登录。

（2）单一文件系统。这有两方面含义。首先，在机群系统中，有一些对整个机群所有结点都相同的软件，没有必要在每一个结点上重复安装。其次，尽管执行并行作业时要求每个结点都可以访问到这些软件，但它们在整个机群系统中应该只有一个备份。

（3）单一作业管理系统。用户可以透明地从任一结点提交作业，作业可以以批处理、交互或并行的方式被调度执行。

【题12.5】 答：按照机群系统中各结点的功能不同，可以分为以下3类。

（1）计算结点，用于完成计算任务。

（2）管理/登录结点，它们是外部设备和机群系统之间连接的桥梁，任何用户和系统的管理员都只能通过此类结点才能登录到系统中。此外，管理/登录结点还应具有管理和作业提交等功能。

(3) I/O结点,作为NFS文件系统的主结点,I/O结点一般由存储设备、网络文件系统(NFS)等组成,外挂磁盘阵列或者连接其他存储设备,负责文件的I/O操作,其他结点访问存储设备的请求都要通过I/O结点完成。

这3类结点所需的具体硬件配置也不相同。计算结点需要提供很强的计算能力,对于某些应用而言特别需要强大的浮点计算能力。此外,计算结点还应提供适量的内存,使运算时数据能完全驻留在物理内存中,并能够支持高速、低延迟的通信网络。而系统对管理/登录结点要求不高,只要采用相对经济的配置就可以了。

【题12.6】 答:根据机群系统的使用目的可以将机群系统分为以下3类。

(1) 高可用性机群。在系统中某些结点出现故障的情况下,仍能继续对外提供服务。它采用冗余机制,当系统中某个结点由于软、硬件故障而失效时,该结点上的任务将在最短的时间内被迁移到机群内另一个具有相同功能与结构的结点上继续执行。这样,对于用户而言,系统可以一直为其提供服务。

(2) 负载均衡机群。提供与结点个数成正比的负载能力,这就要求机群能够根据系统中各个结点的负载情况实时地进行任务分配。这种机群很适合大规模网络应用、大工作量的串行或批处理作业。

(3) 高性能计算机群。降低高性能计算的成本,通过高速的商用互连网络,将数十个乃至上千台PC或工作站连接在一起,可以提供接近甚至超过传统并行计算机系统的计算能力,但其价格却仅是具有相同计算能力的传统并行计算机系统的几十分之一。

【题12.7】 答:专用机群是为代替传统的大中型计算机或巨型计算机而设计的,装置比较紧凑,一般都装在比较小的机架内,放在机房中使用,因此它的吞吐率较高,响应时间也较短。专用机群的结点往往是同构的,一般采用集中控制,由一个(或一组)管理员统一管理,而且用户一般需要通过一台终端机来访问它。

企业机群是为了充分利用各个结点的空闲资源而设计的,因此其各个结点分散安放,并不需要安装在同一个房间,甚至不需要安排在同一幢楼中。各结点一般通过标准的LAN或WAN互连,通信开销较大、延迟较长。企业机群的各个结点一般是异构的,并由不同的个人拥有,这样机群管理者只能对各个结点进行有限的管理,结点拥有者可以随意地进行关机、重新配置或者升级,而且对一个结点而言,它的拥有者的任务应该具有最高优先级,高于企业的其他用户。

第 13 章　阵列处理机

13.1　基本要求与难点

13.1.1　基本要求

(1) 掌握有关阵列处理机的基本概念。

(2) 掌握阵列处理机的两个操作模型及其特点。

(3) 掌握阵列处理机的两种基本结构：分布式存储器的阵列机，共享存储器的阵列处理机。

(4) 理解 Illiac Ⅳ阵列处理机和 BSP 计算机的结构和特点。

(5) 掌握 Illiac Ⅳ阵列的 64 个处理单元的连接方式。

(6) 了解阵列处理机的并行算法举例。

13.1.2　难点

(1) 阵列处理机的两种基本结构。

(2) IlliacⅣ阵列处理机和 BSP 计算机的结构和特点。

13.2　知识要点

13.2.1　阵列处理机的操作模型和特点

1. 阵列处理机的操作模型

阵列处理机的操作模型如图 13.1 所示。它是用一个控制部件 CU 同时管理多个处理单元 PE。CU 对指令进行译码，并把指令播送到各处理单元。所有处理单元均被动地接收并执行从控制部件广播来的同一条指令，但它们所操作的对象却是不同的数据。

阵列处理机的操作模型可用五元组表示：

$$阵列处理机=(N,C,I,M,R)$$

其中：

(1) N 为机器的处理单元(PE)数。

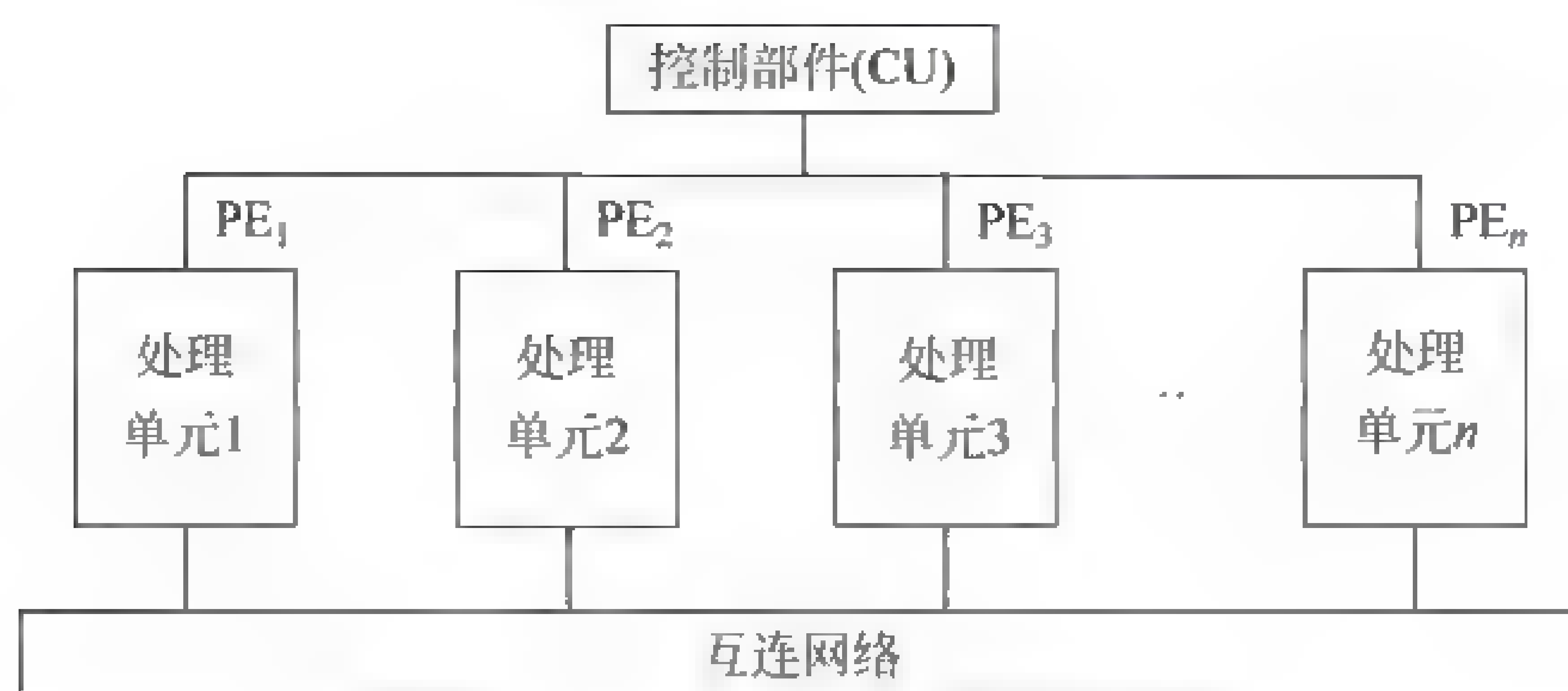


图 13.1　阵列处理机的操作模型

(2) C 为控制部件 CU 直接执行的指令集，包括标量指令和程序流控制指令。

(3) I 为由 CU 广播至所有 PE 进行并行执行的指令集，包括算术运算、逻辑运算、数据寻径、屏蔽以及其他由每个 PE 对它的数据所执行的局部操作。

(4) M 为屏蔽方案集，其中，每种屏蔽将所有 PE 划分成允许操作和禁止操作两种工作模式。

(5) R 为数据寻径功能集，说明互连网络中 PE 间通信所需要的各种设置模式。

2. 阵列处理机的特点

阵列处理机的特点如下。

(1) 阵列机是以单指令流多数据流方式工作的。

(2) 阵列机是通过设置多个相同的处理单元来开发并行性，它是利用并行性中的同时性，而不是并发性。所有处理单元必须同时进行相同的操作。这与利用时间重叠的向量流水处理机是不一样的。

(3) 阵列机是以某一类算法为背景的专用计算机。这是由于阵列机中通常都采用简单、规整的互连网络来实现处理单元间的连接操作，从而限定了它所适用的求解算法类别。

(4) 阵列机的研究必须与并行算法的研究密切结合，以使得能充分发挥它的处理能力。

(5) 阵列机的控制器实质上是一台标量处理机，而为了完成 I/O 操作以及操作系统的管理，尚需一个前端机。因此实际的阵列机系统是由上述三部分构成的一个异构型多处理机系统。

13.2.2　阵列处理机的基本结构

1. 分布式存储器的阵列机

分布式存储器的阵列机中有多个相同的处理单元 PE，每个 PE 有各自的本地存储器 LM。PE 之间通过数据寻径网络以一定方式互相连接。它们在阵列控制部件的统一指挥下，实现并行操作。由于通过控制部件的是单指令流，所以指令的执行顺序还是和单处理机一样，基本上是串行进行。程序和数据是通过主机装入控制存储器。

指令送到控制部件进行译码。如果是标量指令，则直接由标量处理机执行。如果是向量指令，则阵列控制部件通过广播总线将它广播到所有 PE 中去并行地执行。

执行程序所需的数据集经划分后通过数据总线分布存放到各 PE 的本地存储器 LM。各 PE 之间通过数据寻径网络互连，实现 PE 间的通信，如移数、置换和其他寻径操作。控制部件通过执行程序来控制数据寻径网络。

PE 的同步是在控制部件的控制下由硬件实现。我们可以做到让所有 PE 在同一个周期执行同一条指令，也可以做到通过采用屏蔽逻辑来控制某些 PE 在指定的指令周期是否参与执行。Illiac Ⅳ 是这种结构的 SIMD 阵列处理机，它由 64 个带本地存储器的 PE 组成，PE 间通过 8×8 环绕连接网络实现互连。

2. 共享存储器的阵列机

共享存储器的阵列处理机中，共享的多体并行存储器 SM 通过对准网络与各处理单元 PE 相连。存储模块的数目等于或略大于处理单元的数目。为了减少存储器访问冲突，必须将数据合理地分配到各存储器模块中。通过灵活高速的对准网络，使存储器与处理单元之间的数据传送在大多数向量运算中都能以存储器的最高频率进行。这种共享存储器模块在处理单元数目不太大的情况下是很理想的。例如，美国宝来公司的 BSP 计算机 (Burroughs Scienticfic Processor) 采用了这种结构。16 个 PE 通过一个 16×17 的对准网络访问 17 个共享存储器模块。存储器模块数与 PE 个数是互质的，可以实现无冲突并行访问存储器。

所有阵列指令都必须使用长度为 n 的向量操作数，其中，n 为 PE 的个数。SIMD 指令与流水向量处理机的指令类似，不同之处是用多个 PE 的空间并行性代替了流水线的时间并行性。

上述阵列处理机的所有 I/O 操作都是由主机处理的。在主机和阵列控制部件之间有一个专用的控制存储器，用于存放程序和数据。

13.2.3　阵列处理机实例

1. 实例 1：Illiac Ⅳ 阵列处理机

Illiac Ⅳ 阵列处理机是美国宝来公司和伊利诺大学合作研制生产的机器，它于 1972 年问世，是最早的阵列处理机。

Illiac Ⅳ 实际上是一个由 3 种类型处理机联合组成的多机系统。这 3 个处理机是：

(1) 专门用于数组运算的处理单元阵列。

(2) 阵列控制器(CU)，它既是处理单元阵列的控制部分，又可以看作一台相对独立的小型标量处理机。

(3) 一台标准的 B6700 计算机，担负 Illiac Ⅳ 输入输出系统和操作系统管理功能。

1) Illiac Ⅳ 阵列

Illiac Ⅳ 阵列由 64 个处理单元(PE)、64 个本地存储器(PEM)和存储器逻辑部件(MLU)组成。把每个 PE 和 PEM 对看成是一个处理部件 PU，则这个阵列的 64 个处理部件 $PU_0 \sim PU_{63}$ 排列成一个 8×8 方阵。每一个 PU_i 只和其左、右、上、下的 4 个近邻 PU_{i-1}(mod64)、PU_{i+1}(mod64)、PU_{i-8}(mod64)、PU_{i+8}(mod64) 有直接连接。按此规则，上下方向

上同一列的PU两端相连成一个环，左右方向上每一行的右端PU与下一行的左端PU相连，并且最下面一行的右端PU与最上面一行的左端PU相连，从而构成了一个闭合的螺线形状。因此，Illiac Ⅳ的阵列结构又称为闭合螺线阵列。这种连接方式既便于一维长向量的处理，又便于二维数组运算，以缩短处理单元之间的路径距离。步距不等于±1或±8的任意处理单元间通信可用软件方法寻找最短路径进行，其最短距离都不会超过7步。

每一个处理单元有一个自己的本地存储器PEM。64个PEM联合组成阵列存储器，存放数据和指令。整个阵列存储器可以接收控制器的访问，读出8个字的信息到它的缓冲器中，也可经过1024位的总线与I/O开关相连。但是每一个处理单元只能访问自己的本地存储器PEM。分布在各PEM中的公共数据只能在读出到控制器CU后，再经公共数据总线广播到64个处理单元中。这不但节省了存储空间，而且允许公共数据的存取与其他操作在时间上重叠。这样，阵列存储器就如同一个二维访问的存储器。

2) 阵列控制器

阵列控制器CU实际上是一台小型计算机。它除了对阵列的处理单元进行控制以外，还能利用本身的内部资源执行一整套指令，用以完成标量操作。而且这些操作可以与各PE的数组操作并行进行。

阵列控制器的功能如下。

(1) 对指令流进行控制和译码，包括执行一整套标量指令；

(2) 向各处理单元发出执行数组操作指令所需的控制信号；

(3) 产生并向所有处理单元广播公共的地址部分；

(4) 产生并向所有处理单元广播公共的数据；

(5) 接收和处理由各PE计算出错、系统I/O操作以及B6700所产生的陷阱中断信号。

3) 输入输出系统

IlliacⅣ输入输出系统由磁盘文件系统DFS、I/O分系统和B6700管理计算机组成。

4) B6700管理计算机

B6700的作用是：管理全部系统资源，完成用户程序的编译或汇编，为Illiac Ⅳ进行作业调度、存储分配、产生I/O控制描述字送至CDC、处理中断、提供操作系统所具备的其他服务等。

2. 实例2：BSP计算机

BSP计算机是由美国宝来公司和伊利诺依大学于1979年制造的。它是共享存储器结构的SIMD计算机的典型代表。其最高处理性能为每秒5千万次浮点运算。

BSP采用了全面的并行化措施。它不是依靠提高时钟周期频率来提高性能，而是依靠并行性。它的处理器和存储器的时钟周期都是160ns，其时钟频率在巨型处理机中是比较低的，但它依靠重复设置的16个处理单元，仍能获得与Cray 1流水线处理机相当的向量处理速度。

BSP不是一台独立运行的计算机，而是附属于系统管理计算机B7700/B7800的后端处理机。BSP承担主要的计算任务，而系统管理机则负责进行BSP程序的向量化编译和连接、与远程终端及网络的数据通信、外围设备管理等，大多数的BSP作业调度和操作系统活动也是在系统管理机上完成的。

1) BSP 处理机

并行处理机包含 16 个算术单元 AE、由 17 个存储体组成的一个无冲突访问的并行存储器和两套对准网络(分别为入口和出口对准网络)。它们形成了一条 5 级的数据流水线。这 5 级的功能依次是：

(1) 由 17 个存储器输出端口并行读出 16 个操作数；

(2) 经对准网络 NW_1 将 16 个操作数重新排列，形成 16 个算术单元所需要的顺序；

(3) 将排列好的 16 个操作数送到 16 个算术单元进行处理；

(4) 所得的 16 个结果经对准网络 NW_2 重新排列成在 17 个存储体中存储所需要的次序；

(5) 写入并行存储器。

该流水线使连续几条向量指令能在时间上重叠起来执行。

两套对准网络的作用分别是在读或写并行存储器时，使并行存储器中为保证无冲突访问而错开存放的操作数顺序能够与算术单元并行处理要求的正常顺序协调一致。整个流水线由统一的指令译码和控制部件进行控制。

这种流水线对提高系统处理效率有很大作用。第一，有效地实现了处理单元、存储器和互连网络在时间上重叠工作，在理想情况下能取得带宽的完全匹配。第二，可把大于 16 的任意长度的向量按 16 个分量的标准长度分为若干段，依次在时间上重叠起来进行处理。第三，实现不同向量指令的重叠执行。

进行向量运算的数据保存在由 17 个存储体组成的并行存储器中，每个存储体的容量可达 512K 字，存储周期为 160ns。数据在存储体和 AE 之间以每秒 100MB 的速率进行传输。17 个存储体组成一个无冲突访问存储器，它允许对任意长度以及跳距不是 17 的倍数的向量实现无冲突存取。

对准网络包含完全交叉开关以及用来实现数据从一个源广播至几个目的地的硬件，还包含当几个源寻找同一个目的地时能分解冲突的硬件。在算术单元阵列和并行存储器的存储体之间具备通用的互连特性，而存储体和对准网络的组合功能则提供了并行存储器的无冲突访问的能力。

2) BSP 并行存储器

BSP 并行存储器由 17 个存储体组成。16 个 AE 在每个存储周期对并行存储器存/取 16 个字。16 个 AE 执行一次浮点加/减/乘法运算需要 32 个操作数，故而需要用两个周期从并行存储器中获得，而算术单元的浮点加、减、乘运算都能在两个周期内完成。因此，并行存储器的带宽同算术单元的浮点运算的带宽保持完全平衡，从而可将并行存储器的存取操作同 16 个算术单元的运算操作按时间重叠进行流水处理。

BSP 并行存储器的一个独特的性能就是它可以实现无冲突访问。对于数组，它可以按行、列、对角线等进行访问而不会产生冲突。实现无冲突访问的硬件支持包括：质数个存储器端口(存储体数是质数 17)；存储端口和 AE 之间的交叉开关(对准网络)；以及特殊的存储器地址生成机构。

在 BSP 中，当对数组的同一行、同一列、主对角线、次对角线的元素并行地读取时都不会发生访问存储体的冲突，因为它们在不同的存储体中。

BSP 有 16 个算术单元，即 $N=16$，存储体数 $M=17$，因此，并行存储器的每个访问周期

总有一个存储体未被利用,使得并行存储器的空间利用率和存储器带宽都浪费了1/17,但是这种损失却换来了对存储器的无冲突访问。

13.2.4 阵列处理机的并行算法举例

1. 有限差分问题

有限差分方法是求解场方程的一种有效方法。它把一个有规则的网格覆盖在整个场域上,用网格点上的变量值写出差分方程组以代替场方程来进行计算。

IlliacⅣ在计算时,是把内部网格点分配给各个处理单元。因此,其计算过程可以并行地完成,从而大幅度地提高处理速度。由于实际问题所遇到的内部网格点数目往往是很大的,因此需要将其分成许多子网络,然后才能在Illiac Ⅳ上求解。

2. 矩阵加

在阵列处理机上解决矩阵加法问题是最简单的一维情况。考虑两个8×8的矩阵$\boldsymbol{A}$和$\boldsymbol{B}$的相加,所得结果矩阵$\boldsymbol{C}$也是一个8×8的矩阵。把$\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$中位于相应位置的分量存放在同一PEM内,假设$\boldsymbol{A}$的分量在全部64个PEM中存放的单元地址都是α,$\boldsymbol{B}$的全部分量的地址都是$\alpha+1$,$\boldsymbol{C}$的全部分量的地址都是$\alpha+2$。这样,只需用下列3条Illiac Ⅳ的汇编指令就可以实现矩阵相加。

```
LDA     ALPHA        ;全部A的分量由PEMi送PEi的累加器RGAi
ADRN    ALPHA+1      ;全部B的分量与(RGAi)进行浮点加,结果送RGAi
STA     ALPHA+2      ;全部(RGAi)由PEi送PEMi的α+2单元
```

3. 矩阵乘

矩阵乘是二维数组运算,比矩阵加要复杂许多。设$\boldsymbol{A}$、$\boldsymbol{B}$和$\boldsymbol{C}$为3个8×8的二维矩阵。若给定$\boldsymbol{A}$和$\boldsymbol{B}$,则$\boldsymbol{C}=\boldsymbol{A}*\boldsymbol{B}$的64个分量可利用下列公式计算。

$$c_{ij}=\sum_{k=0}^{7}a_{ik}b_{kj},\quad 0\leqslant i,j\leqslant 7$$

如果在SIMD阵列处理机上求解这个问题,可以利用8个处理单元并行计算i或者j的8个循环,从而消去一重循环。例如,并行计算j的8个循环,则i、k循环照旧。这时可执行下列FORTRAN程序:

```
      DO 10 i = 0,7
      C(i,j) = 0
      DO 10 k = 0,7
10    C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

这样,便可把速度提高到原来的8倍,即每个处理单元的计算时间缩短为64次乘加时间。

如果把Illiac Ⅳ的64个处理单元全部利用起来并行计算,即把k循环的运算也改为并行,则可进一步提高速度。但这需要重新在阵列存储器中恰当地分配数据,还要能使8个中

间积 $A(i,k)\times B(k,j)$能并行相加($0\leq k\leq 7$)。这就要用到下面的累加和并行算法。不过，即便如此，就 k 的并行来说，速度的提高不是 8 倍，而是 $8/\log_2 8\approx 2.7$ 倍。

4. 累加和

这是一个将 N 个数的顺序相加转变为并行相加的问题。这个计算任务要用到处理单元中的活动标志位。只有处于活动状态的处理单元，才能执行相应的操作。

习　题

1. 概念题

【题 13.1】 解释以下名词

阵列处理机　　阵列控制器　　分布式存储器的阵列机　　共享存储器的阵列机

2. 填空题

【题 13.2】 阵列处理机又称为________。其采用的主要技术手段是硬件上采用________的方法来实现并行性。

【题 13.3】 阵列处理机操作模型用五元组表示：阵列处理机＝(N,C,I,M,R)，其中，N 表示机器的________个数，I 表示由控制部件 CU 广播至所有 PE 进行并行执行的________，M 为________，其中每种屏蔽将所有 PE 划分成________和________两种工作模式。

【题 13.4】 多处理机与并行处理机的本质差别在于并行性级别的不同：多处理机实现________的并行，而并行处理机则实现同一指令多数据流的________的并行。

【题 13.5】 阵列处理机的基本结构可分为________的阵列机和________的阵列机两大类。

【题 13.6】 Illiac Ⅳ阵列处理机处理单元 PE 之间互连用互连函数表示为________。

【题 13.7】 Illiac Ⅳ阵列机是一个由以下 3 种类型处理机联合组成的多机系统：①处理部件阵列 PU 专门用于________运算；②阵列控制器 CU 是一台相对独立的小型________处理机；③B6700 计算机担负 I/O 系统和操作系统管理功能。

【题 13.8】 Illiac Ⅳ阵列 PU 由 64 个________、64 个________和存储器逻辑部件(MLU)组成。

【题 13.9】 BSP 计算机采用________存储器结构。它有________个处理单元，并行存储器的存储体个数为________。

【题 13.10】 BSP 处理机由三部分构成：________、________和________。

3. 问答题

【题 13.11】 简述阵列处理机的特点。

【题 13.12】 阵列处理机在系统组成上应包含哪些部分和功能？

【题 13.13】 试分析与比较阵列处理机与向量计算机的相同与不同。

【题 13.14】 简述阵列处理机的分布式存储器与共享存储器的异同。

4. 应用题

【题 13.15】 在集中式主存的阵列处理机中,处理单元数为4,为了使4×4的二维数组A的各元素$a_{ij}(i=0\sim3, j=0\sim3)$在行、列、主/次对角线上均能实现无冲突访问,请填出数组各元素在存储器各分体(分体号从0开始)中的分布情况。假设a_{00}已放在分体号为3,体内地址为$i+0$的位置,如表13.1所示。

表 13.1 数组各元素在存储器各分体中的分布情况

分体号 / 体内地址	0	1	2	3	4
$i+0$				a_{00}	
$i+1$					
$i+2$					
$i+3$					

【题 13.16】 设并行存储器的体数$M=7$(质数),运算单元数$N=6$。考虑下述4×5的数组:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}$$

按BSP地址映像规则,写出这个4×5二维数组在$M=7$、$N=6$的BSP并行存储器中存储的情况。

题 解

1. 概念题

【题 13.1】 解释以下名词

阵列处理机 —— 又称为并行处理机、SIMD计算机。其核心是一个由多个处理单元构成的阵列,用单一的控制部件来控制多个处理单元对各自的数据进行相同的运算和操作。

阵列控制器 —— 阵列控制器实际上是一台计算机,它除了对阵列的处理单元实行控制以外,还能利用本身的内部资源执行一整套指令,用以完成标量操作,且在时间上与各处理单元的数组操作并行进行。

分布式存储器的阵列机 —— 它含有多个同样结构的处理单元PE,通过数据寻径网络以一定方式互相连接。每个PE有各自的本地存储器。在阵列控制部件的统一指挥下,实现并行操作。

共享存储器的阵列机——这类处理机中集中设置存储器。共享的多体并行存储器通过对准网络与各处理单元相连。存储模块的数目等于或略大于处理单元的数目。为了减少存储器访问冲突，必须将数据合理地分配到各存储器模块中。通过灵活高速的对准网络，使存储器与处理单元之间的数据传送在大多数向量运算中都能以存储器的最高频率进行。

2. 填空题

【题 13.2】 答：并行处理机、资源重复

【题 13.3】 答：处理单元、指令集、屏蔽方案集、允许操作、禁止操作

【题 13.4】 答：任务一级、操作一级

【题 13.5】 答：分布式存储器、共享式存储器

【题 13.6】 答：$PM2_{\pm 0}$ 和 $PM2_{\pm 3}$

【题 13.7】 答：数组、标量

【题 13.8】 答：处理单元(PE)、局部存储器(PEM)

【题 13.9】 答：共享、16 、17

【题 13.10】 答：控制处理机、并行处理机、文件存储器

3. 问答题

【题 13.11】 答：

(1) 阵列机是以单指令流多数据流方式工作的。

(2) 阵列机是通过设置多个相同的处理单元来开发并行性，它是利用并行性中的同时性，而不是并发性。所有处理单元必须同时进行相同的操作。

(3) 阵列机是以某一类算法为背景的专用计算机。这是由于阵列机中通常都采用简单、规整的互连网络来实现处理单元间的连接操作，从而限定了它所适用的求解算法类别。

(4) 阵列机的研究必须与并行算法的研究密切结合，以使得能充分发挥它的处理能力。

(5) 阵列机的控制器实质上是一台标量处理机，而为了完成 I/O 操作以及操作系统的管理，尚需一个前端机。因此实际的阵列机系统是由上述三部分构成的一个异构型多处理机系统。

【题 13.12】 答：

(1) 重复设置大量的处理单元用规整灵活的互连网络互连，组成处理单元阵列；

(2) 用专门的并行算法对数组、向量中的元素进行并行处理；

(3) 用一台高性能处理机来进行标量处理和控制互连网络的连接；

(4) 用一台管理处理机来运行系统程序和输入输出任务。

【题 13.13】 答：阵列处理机和向量计算机的相同点是两种计算机都能对大量数据进行向量处理，特别适合于高速数值计算。

不同点是阵列处理机获得高处理速度的主要原因是采用资源重复的并行措施，多个处理单元并行工作；向量计算机依靠的是多功能流水线部件，采用时间重叠提高速度。另一区别是阵列处理机有它的互连网络。

【题 13.14】 答：分布式存储器的阵列机与共享存储器的阵列机的相同点是都存在互连网络。不同点是在共享内存方案中，共享的多体并行存储器通过对准网络与各处理单元

相连。在分布式内存方案中,每个处理单元都有自己的本地存储器,处理单元之间的数据通信通过数据寻径网络完成。

4. 应用题

【题 13.15】

解:数组各元素在各存储器分体中分布情况如表 13.2 所示。

表 13.2 数组各元素在各存储器分体中分布情况

体内地址 \ 分体号	0	1	2	3	4
$i+0$	a_{02}	a_{03}		a_{00}	a_{01}
$i+1$	a_{10}	a_{11}	a_{12}	a_{13}	
$i+2$	a_{23}		a_{20}	a_{21}	a_{22}
$i+3$	a_{31}	a_{32}	a_{33}		a_{30}

【题 13.16】

解:二维数组在 BSP 并行存储器中存储的情况如图 13.2 所示。

数组元素	a_{00}	a_{10}	a_{20}	a_{30}	a_{01}	a_{11}	a_{21}	a_{31}	a_{02}	a_{12}	a_{22}	a_{32}	a_{03}	a_{13}	a_{23}	a_{33}	a_{04}	a_{14}	a_{24}	a_{34}
线性地址 A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
体号 j	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5
体内地址 i	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2	3	3

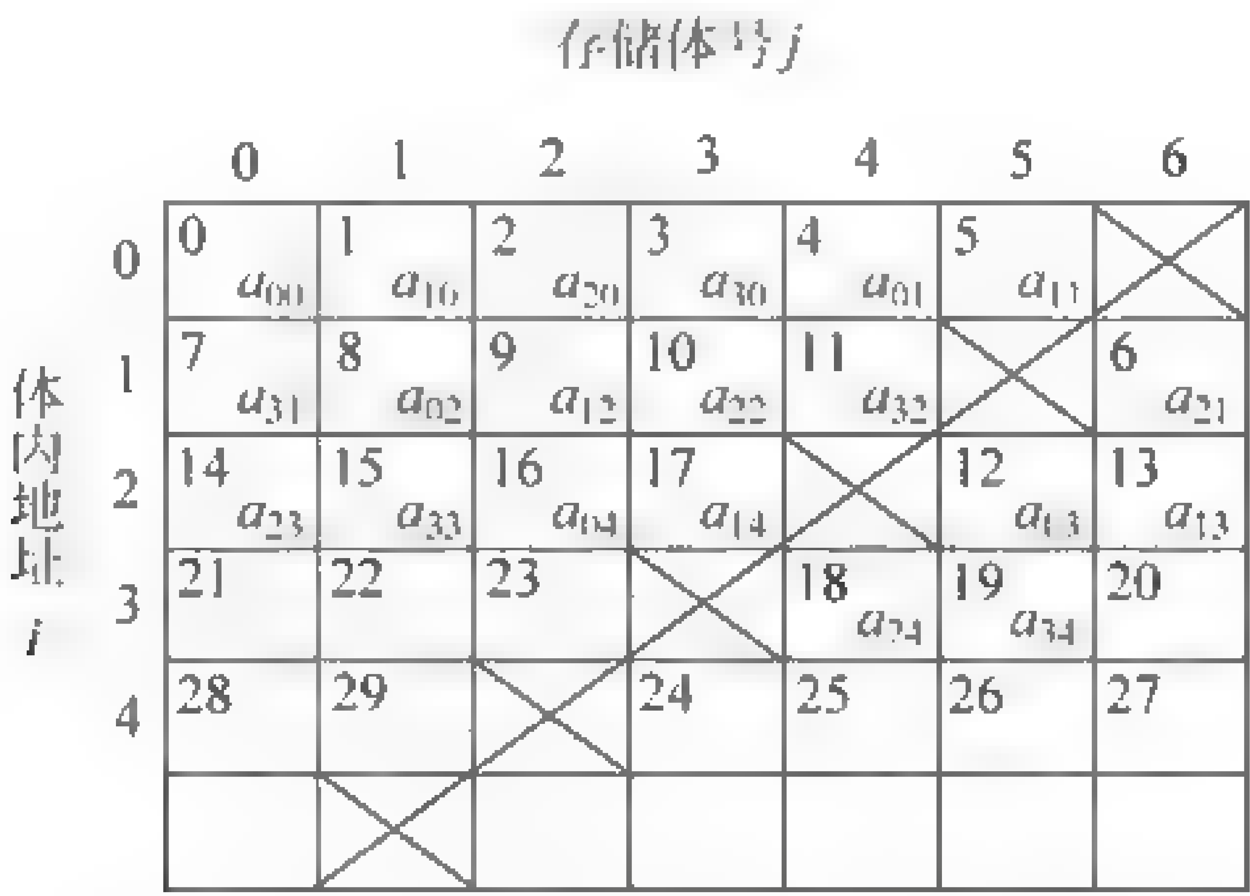


图 13.2 二维数组在 BSP 并行存储器中存储的情况

第14章　数据流计算机

14.1　基本要求与难点

14.1.1　基本要求

（1）掌握有关数据流计算机的基本概念。

（2）掌握数据流计算机的数据驱动原理、指令的执行过程以及数据流计算机的指令结构。

（3）理解用于数据流程序图的常用结点，并能熟练地画出数据流程序图。

（4）掌握静态数据流计算机的结构和工作原理。

（5）掌握动态数据流计算机的结构和工作原理。

（6）了解数据流计算机的优缺点。

14.1.2　难点

（1）画数据流程序图。

（2）动态数据流计算机的结构和工作原理。

14.2　知识要点

14.2.1　数据流计算机的基本原理

1. 数据驱动原理

数据流计算机一般采用数据驱动方式工作，它没有程序计数器，没有常规的变量概念。它的指令是在数据可用性的控制下并行执行的。其基本原理可以归纳为：

（1）当且仅当指令所需要的数据可用时，该指令即可执行。

（2）任何操作都是纯函数操作。

上述的**数据驱动计算**只是数据流计算机中驱动方式的一种。还有一种叫需求驱动计算，也称需求驱动方式。它只在当某一个函数需要用到某一个自变量时才驱动对该自变量的求值操作。前者是一种提前计算的策略，而后者则是按需求值，是一种滞后计算的策略。

目前大多数数据流计算机都是采用数据驱动方式。

2. 数据流计算机中指令的执行过程

在数据流计算机中,用**数据令牌**传送数据并激活指令,用一种有向图来表示数据流程序。一条指令主要由一个操作码、一个或几个操作数、一个或几个后继指令地址组成。后继指令地址用于把本指令的执行结果送往需要这个数据的指令中。

数据驱动具有以下特性。

(1) 异步性:只要指令所需的数据令牌都已到达,指令即可独立地开始执行,而不必关心其他指令及数据的情况。

(2) 并行性:可同时并行执行多条指令。

(3) 函数性:由于不使用共享的数据存储单元,所以数据流程序不会产生诸如改变存储字这样的副作用,也就是说数据流运算是纯函数性的。

(4) 局部性:运算过程中所产生的数据不是用操作数的地址来引用,而是作为数据令牌直接传送,因此数据流运算没有产生长远影响的结果,其运算具有局部性。

3. 数据流计算机的指令结构

图14.1是数据流计算机中指令的结构示意图。其指令主要由两部分组成:操作包、数据令牌。操作包通常由操作码、一个或几个源操作数以及一个或几个后继指令地址组成。后继指令地址用于组成新的数据令牌,以便把本条指令的运算结果送往需要它的目标指令。

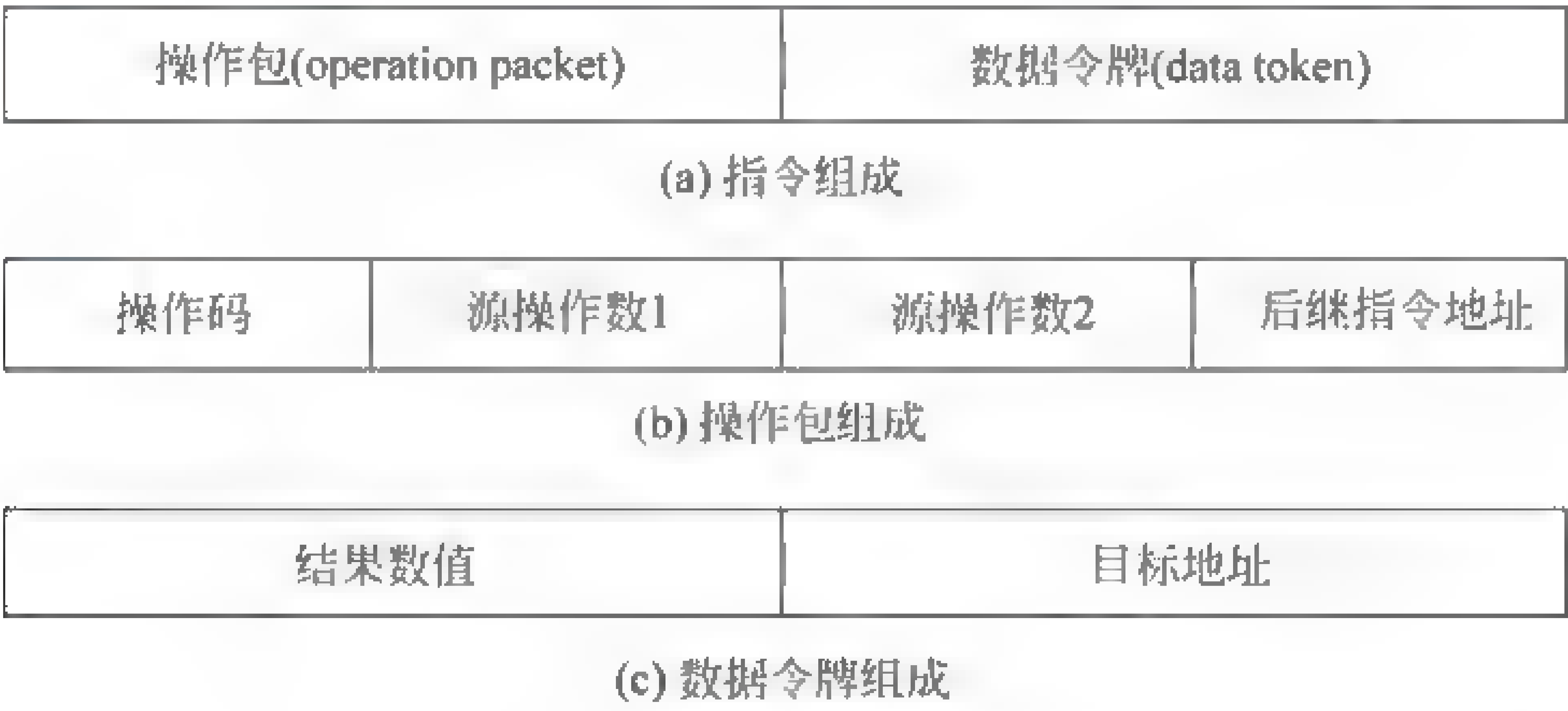


图 14.1 数据流计算机指令的组成

数据令牌通常由结果数值和目标地址等组成,如图14.1(c)所示。其中,结果数值就是上条指令的运算结果,而目标地址则直接取自上条指令的后继指令地址。如果一条指令的执行结果要送往几个目标地址,则要分别形成几个数据令牌。

14.2.2 数据流程序图和数据流语言

数据流计算机的程序是用数据流语言编写的。最基本的数据流语言是数据流计算机的机器语言,即数据流程序图。

1. 数据流程序图

数据流程序图是一种特殊的有向图，由多个结点以及连接这些结点的有向弧构成。结点用圆圈、三角形、菱形、椭圆等形状表示，其中的符号表示进行什么操作。弧代表结点之间的关系及令牌流向。这种表示法也称为结点分支线表示法。

J. B. Dennis 和 J. E. Rumbaugh 等提出了用于数据流程序图的各种符号(即结点)，并规定了相应的操作执行规则。现对主要的一些结点简要介绍如下。

(1) 复制操作结点：可以是数据的复制，也可以是控制量(布尔量)的复制。数据复制的箭头是实心的。如果要表示控制量的复制，只要把箭头改为空心的即可。复制操作结点有时也称连接操作结点。

(2) 运算操作结点：主要包括常用的加、减、乘、除、乘方、开方等算术运算及与、或、非、异或、或非等布尔逻辑运算。激发后输出带相应操作结果的令牌。

(3) 常数发生器结点：没有输入端，只有一条输出分支。它用于产生常数，激发后输出带常数的令牌。

(4) 判定操作结点：对输入数据按某种关系进行判断和比较，然后在输出端给出带逻辑值真(T)或假(F)的控制令牌。

(5) 控制类操作结点：控制类操作结点的激发条件需要加入布尔控制端。

除了上面所介绍的结点分支线方法之外，还有一种表示数据流程序图的方法，称为活动模片表示法。在采用活动模片表示法的数据流程序图中，组成数据流程序图的基本单位是活动模片。每个活动模片相当于结点分支线表示法中的一个或几个操作结点。一个活动模片通常由 4 个字段组成，如图 14.2 所示。

活动片标记	操作码	操作数1	操作数2	目标活动片/部位号

图 14.2　活动模片的结构

活动模片实际上是结点在数据流机器内部具体实现时的存储器映像。比较而言，活动模片表示法更接近于常规计算机中的机器代码，能够由数据流计算机硬件直接解释执行。但结点分支线表示法更接近于一般的高级语言，具有可读性好、直观等优点，在程序设计中被广泛采用。

2. 数据流语言及其性质

如前所述，**数据流程序图**实际上是数据流计算机的机器语言。其优点是直观易懂，但编程效率很低，难以被一般用户所接受。因此需要有适合于数据流计算机的高级语言。

目前，数据流语言的研究还很不成熟，还没有形成像传统高级语言那样的规范版本。有的数据流机就是用传统的命令式语言作为数据流计算机的高级语言，用专门的编译程序将用该高级语言编写的程序转换成数据流程序图。但效果不是很好。所以研究和发展新的、适合于数据流机制的高级语言是非常重要的。

已经出现的比较适合于表述数据流程序的高级语言有以下两种。

(1) 单赋值语言。包括美国加州大学 Irvine 分校研制的 ID 语言，美国 MIT 实验室的

VAL,法国的LAU语言,英国曼切斯特大学的SISAL语言等。

(2) 函数类语言。比较著名的有美国犹他大学研制的FP语言。

14.2.3　数据流计算机结构

按照对数据令牌处理方法的不同,可把数据流计算机分为两种:静态数据流计算机和动态数据流计算机。

1. 静态数据流计算机

静态数据流计算机的结构如图14.3所示。

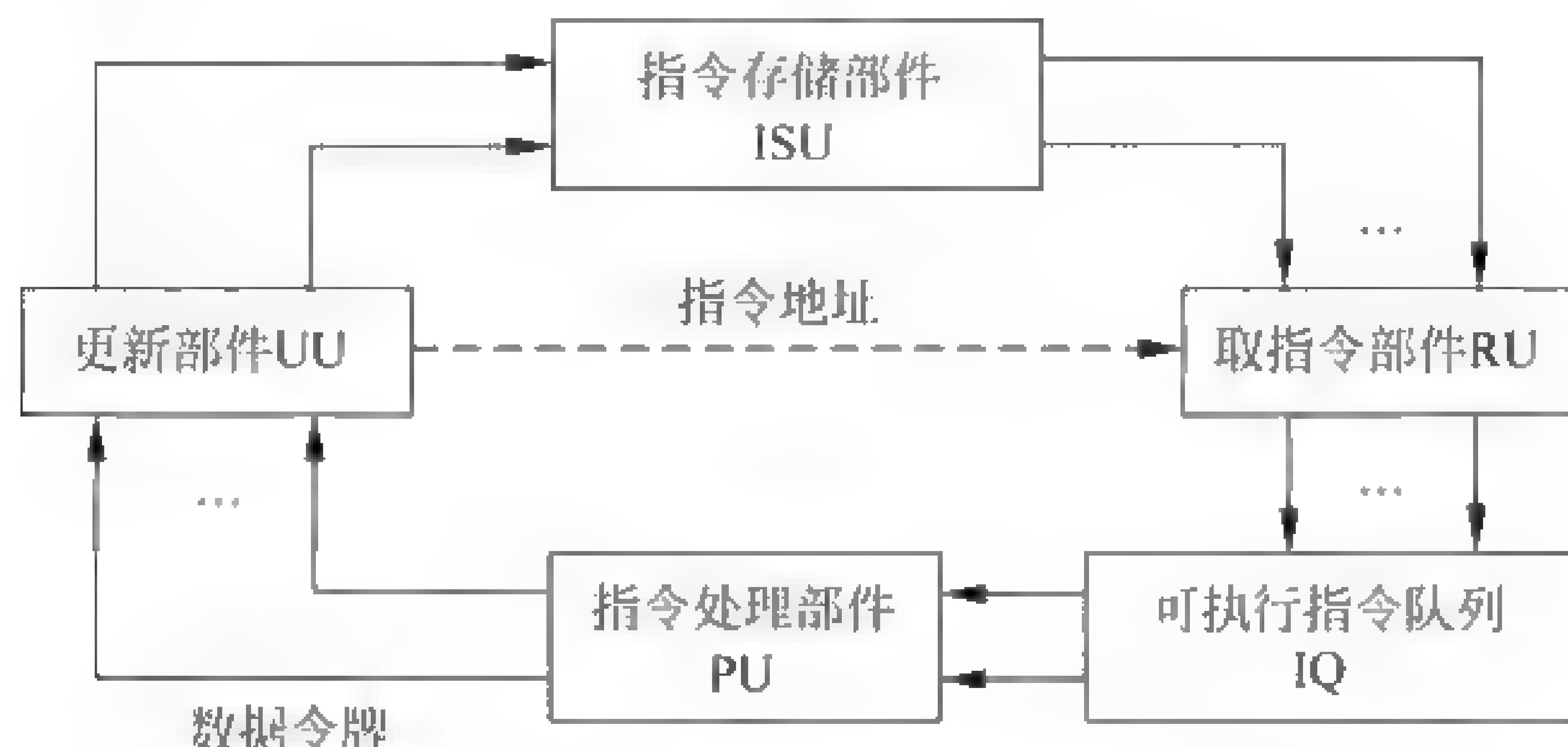


图14.3　静态数据流计算机的结构

要执行的数据流程序存放在指令存储部件ISU中。数据令牌开始是存放在更新部件UU的输入缓冲器中,UU根据数据令牌所携带的目标地址,把令牌中的操作数送往ISU中相应指令的有关位置。当一条指令所需要的数据令牌全部到达后,这条指令将被激活。UU会把这样的指令的地址传送给RU。RU把这些指令取出,送到可执行指令队列IQ中。若PU中有空闲的处理机,分派程序将按先后次序把处理机分配给等待执行的指令,使它们并行执行。指令执行后,其结果将与指令中给出的后继指令地址结合,组成新的数据令牌,并被送入UU的输入缓冲器。如此反复。

在静态数据流计算机中,数据令牌是沿数据流程序图中的有向分支流向操作结点的。当一个结点的所有输入分支线上的数据令牌都到达,且输出分支线上没有数据令牌时,就可以执行该结点的操作。这称为点火。另外还规定,在任何一个时钟节拍内,在任何一条分支线上只允许传送一个数据令牌,这样做的好处是不必在数据令牌中附加标号,使得静态数据流计算机的结构比较简单。但对程序并行性的支持不够。

2. 动态数据流计算机

在**动态数据流计算机**中,数据令牌可以带有标记,称为**带标记的数据令牌**。数据令牌的标记唯一地确定了令牌的状态及其他相关信息。因此,当数据令牌在数据流程序图的有向分支线上流动时,同一条分支线上可以同时有几个数据令牌在移动。

典型的动态数据流计算机的基本结构如图14.4所示。

匹配部件将处理部件中各处理单元送来的结果数据令牌赋予相应的标记,并将流向同

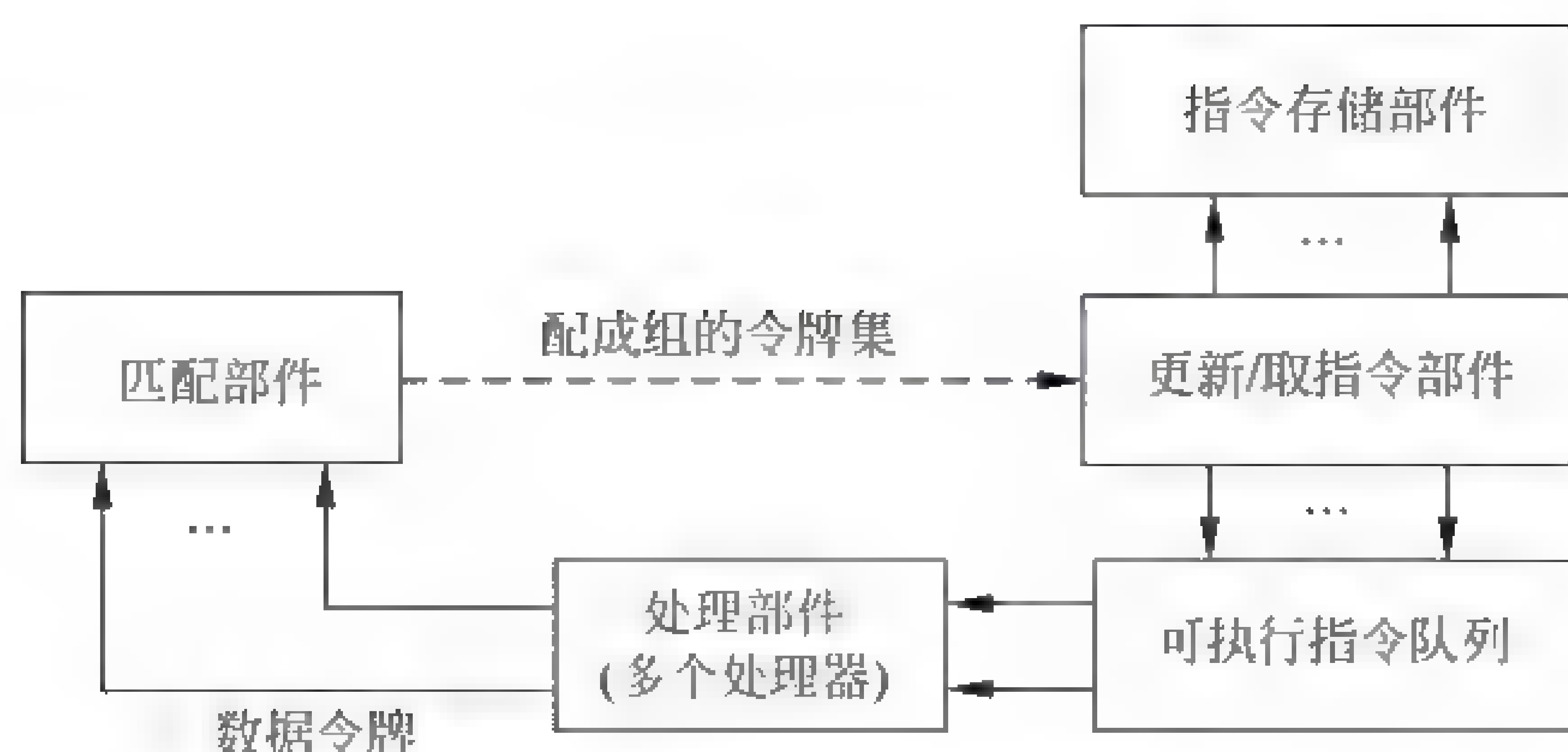


图 14.4 动态数据流计算机的结构

一指令(标记相符)的数据令牌匹配成对或者组。当一条指令所需要的数据令牌全部到齐后,该指令将被激活。已匹配的数据令牌组被送往更新/取指令部件,由该部件把需要它们的指令从指令存储部件中取出,并把该指令与数据令牌组携带的操作数组成一个操作包送入可执行指令队列,然后由处理部件执行。若匹配失败,即指令所需要的数据令牌没有全部到齐,则把已到达的数据令牌暂存在匹配部件的缓冲存储器中,供下次匹配时再使用。匹配缓冲存储器通常是一个相联存储器。

动态数据流计算机中给数据令牌赋予标记的方法有利于最大限度地开发程序中的并行性。如果程序是循环的,则标记方法可以区分出不同层次的迭代执行,实现它们的并行计算。

动态数据流计算机从结构上可以分为两大类。一类是以 MIT 为代表的网络型结构数据流计算机,另一类是以 Manchester 为代表的环状结构数据流计算机。下面介绍其结构特点和工作原理等。

1) 网络型结构动态数据流计算机

MIT 动态数据流计算机是网络型结构的典型代表。它由 N 个处理单元 PE 和一个 $N\times N$ 的包交换开关网络组成。PE 之间通过这个开关网络进行信息交换。每个 PE 基本上就是一台完整的处理机,有自己的存储器、算术逻辑运算部件、标记匹配部件等。

“等待–匹配”部件将已经接收到的令牌暂时保存在它的缓冲器中,等待后续令牌的到来。当所有所需的令牌均已到达,且标记匹配,就将它们送到取指令部件的缓冲器中。根据标记中的信息,从程序/数据存储器中取出相应的指令,然后形成包含操作码、操作数和目的地址的操作包,送到执行部件去执行。执行部件中有一个浮点和逻辑运算部件以及一个用于确定下一个操作和目的地该用哪个 PE 的硬件。

I 结构存储器是一个带标记的专用存储器,用于存放类似于数组的数据结构。如果操作数是结构数据,则该数据从 I 结构存储器取得。I 结构的每一个元素都有一位标志。当读出时,若该元素的标志为 0,就表示其值尚未产生,则自动推迟读操作。使用这种 I 结构存储器可以避免过多的数组复制操作,节省大量的存储空间和辅助操作开销。

在运算部件或存储器产生的结果配上新的标记和目的地 PE 号后,被送往输出口。通过输出口经开关网络送往目标 PE。如果目的地 PE 就是本 PE,就直接送入本 PE 的输入口,而不经过开关网络。为了避免在 PE 各部件中多个通路传输数据时发生冲突、阻塞或死锁,各个部件都设有相应的缓冲器。

PE中没有程序计数器，但有一个存放已激活指令的列表，使已激活指令的执行是无序的。

$N\times N$ 的开关网络是用 $\frac{N}{2}\log_2 N$ 个 2×2 的开关单元组成的 $\log_2 N$ 级网络。基本形式与多级互连网络类似。在每个开关入口有一个异步控制器，用来控制不等长的包交换；并且还有相应的输入缓冲器；开关出口处有仲裁线路，用来解决可能出现的路径冲突。

2）环状结构动态数据流计算机

Manchester动态数据流计算机是环状结构的典型代表，其结构如图14.5所示，它由5个功能部件组成，按顺时针方向进行连接，形成一个环状流水线。这5个部件是：交换开关网络，令牌队列，匹配部件，结点存储器，处理部件。这种结构允许多个令牌以先进先出的队列形式同时存在于数据流程序图的一个弧上。该数据流计算机采用令牌包通信。

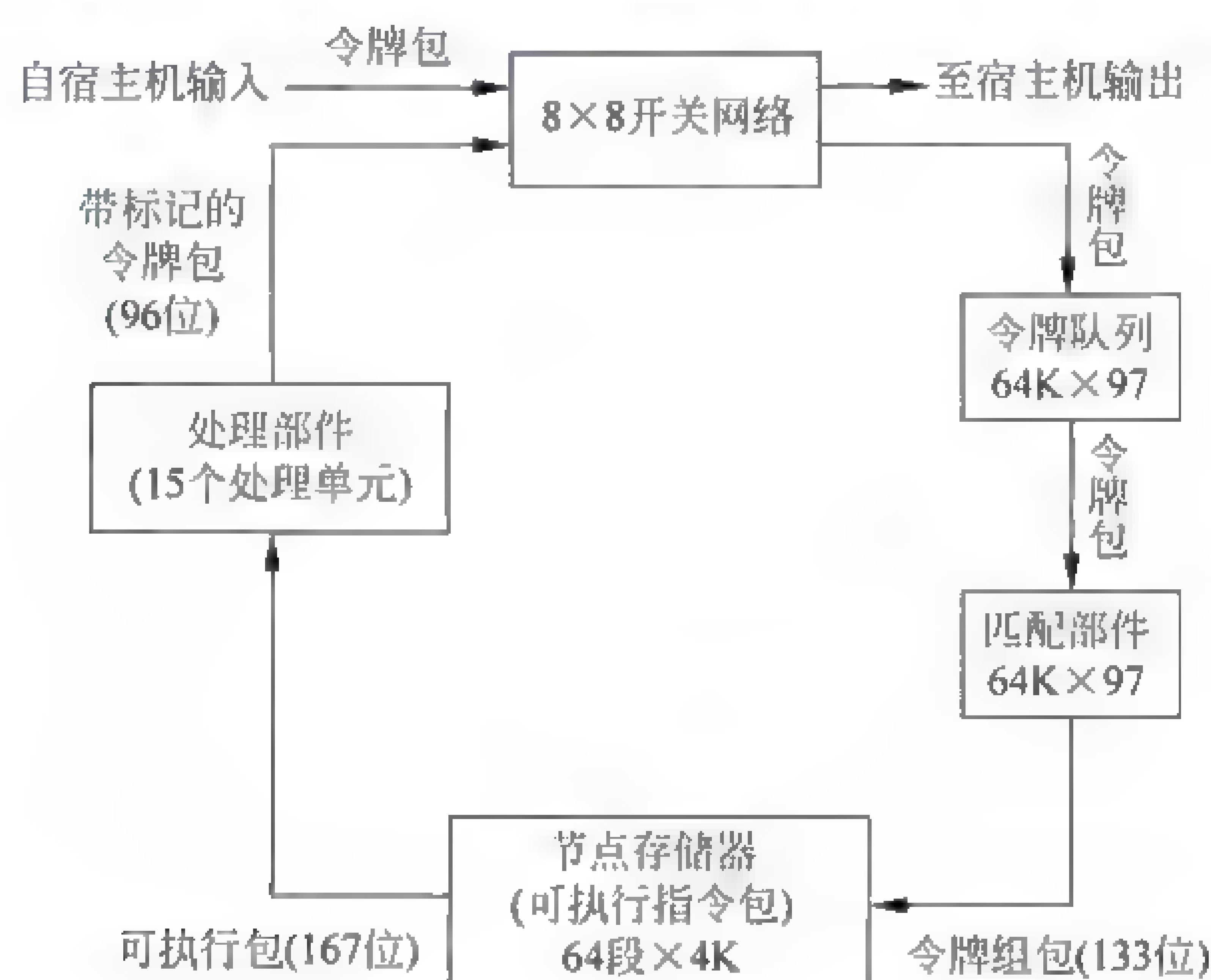


图14.5 Manchester动态数据流计算机的结构

在Manchester动态数据流计算机中，令牌主要由数值、标记以及目标结点地址等几部分组成，而指令则是由操作码、操作数、标记、数据令牌的目的指令（两个）等组成。标记中包含三部分信息：令牌所属进程的标识符，令牌所在的数据流程序图中的弧，代表弧上第几个令牌的迭代序号。

处理部件由15个功能相同的处理单元PE组成，这些PE可并行地执行指令。所执行的指令包括定点运算、浮点运算、数据传送、打标记等。每个PE都有输入缓冲器和输出缓冲器。

从开关网络输入的数据令牌组成一个令牌包存入令牌队列，令牌队列按先进先出的方式工作。匹配部件按照令牌的标记对令牌进行匹配，即把从令牌队列中送来的数据令牌与匹配部件中已经存在的令牌进行比较，看是否具有相同的标记。当一个结点（指令）所需的令牌全部到达后，该结点即被激活。这时，将数据令牌和匹配标记合并成一个令牌组包，送往结点（指令）存储器。该存储器存放数据流程序。根据目的结点说明符从结点存储器中取出相应的指令，并与令牌组包携带的操作数组成可执行包，送到处理部件上执行。处理部件执行完后，将结果令牌输出，送入8×8的交换开关网络。

操作结果可根据需要经交换开关网络送往宿主机、外围设备或另一台数据流计算机。Manchester 数据流计算机所具备的高度并行数据驱动能力，使其有很高的处理速度。

Manchester 动态数据流计算机采用高级数据流语言 Lapes 编程，这是一种单赋值的程序设计语言，其语法规则类似于 PASCAL 语言。

14.2.4　数据流计算机的评价

1. 数据流计算机的优点

1）高度的操作并行性

数据流计算机可以实现操作的高度并行性，它不仅能够开发程序中有规则的并行性，而且可以开发程序运行中隐含的并行性。

2）流水线异步操作

与冯·诺依曼型计算机不同，数据流计算机中的指令不是引用操作数的地址，而是直接使用操作数本身，所以数据流计算机实现的是无副作用的纯函数型程序设计方法。

3）与 VLSI 技术相适应

由于数据流计算机结构的基本组成所具有的模块性和均匀性正好与 VLSI 技术相适应。因此，完全有可能充分利用 VLSI 技术，研制出具有很高性能价格比的数据流计算机。

4）有利于提高程序设计效率

数据流计算机使用的是基于纯函数操作的程序设计语言，它完全摒弃了传统编程语言所依赖的变量和变量赋值机制，从而彻底消除了编程中使用全局变量和同名变量所产生的副作用。

2. 数据流计算机的缺点

不过，也有一部分人认为，上述这些数据流技术的优点实际上只有在理论化的数据流计算机模型中才具备，实际的数据流计算机为获得这些优点往往要付出巨大的代价，从而使得实际上的数据流计算机具有许多明显的缺点。也许正是由于这些缺点，才使得到目前为止，数据流计算机发展一直很慢，更不用说与传统的计算机相竞争了。

数据流计算机的缺点如下。

（1）系统开销大；

（2）不能有效利用传统计算机的研究成果；

（3）数据流语言尚不完善。

3. 数据流计算机设计中需解决的问题

综合上面的论述可知，数据流计算机设计中需要解决以下主要技术问题。

（1）研制易于使用、易于用硬件实现的高级数据流语言；

（2）研究程序如何分解，研究如何把程序模块分配给各处理部件的算法；

（3）设计出性能价格比高的信息包交换网络，实现资源冲突的仲裁和数据令牌的分配

等大量通信工作；

(4) 研究智能化的数据驱动机构；

(5) 研究如何在数据流环境中高效率地处理复杂的数据结构；

(6) 研究支持数据流运算的存储系统和存储分配方案；

(7) 在广泛的应用领域里，对数据流计算机的硬件和软件做出性能评价，估计各种系统开销；

(8) 研究数据流计算机的操作系统；

(9) 开发数据流语言的跟踪调试工具。

习　　题

1. 概念题

【题 14.1】 解释以下名词

数据流计算机　　数据驱动计算　　需求驱动计算

数据令牌　　数据流程序图

2. 问答题

【题 14.2】 数据驱动具有哪些特性？

【题 14.3】 简述数据流计算机指令的组成。

【题 14.4】 VAL 语言具有哪些优点？

【题 14.5】 简述静态和动态数据流计算机的主要区别。

【题 14.6】 为什么数据流计算机中通常要设置 I-结构存储器？

【题 14.7】 数据流计算机有哪些优缺点？

3. 应用题

【题 14.8】 用结点分支线方法画出求解

$$x = \sqrt{(a+b) \times d/c - e/d}$$

的数据流程序图。当 $a=4$、$b=8$ 时，画出该数据流程序图的执行过程。

【题 14.9】 用结点分支线方法画出

$$z=(\text{IF } x=10 \text{ THEN } x-y \text{ ELSE } x+y)/y$$

的数据流程序图。

【题 14.10】 用活动模片表示法画出计算

$$x = a \times b + a/b$$

的数据流程序图。

【题 14.11】 利用单功能操作结点实现一般高级语言中的条件语句：

```
if  true  then G1 else G2
```

试画出数据流程序图，其中的 G1 和 G2 都是各自独立的数据流程序图。

【题 14.12】 利用单功能操作结点实现一般高级语言中的循环语句：

```
while  P  do  G
```

或

```
until  P  do  G
```

试画出数据流程序图，其中，P 是循环条件，G 是循环体。

题　　解

1. 概念题

【题 14.1】 解释以下名词

数据流计算机——它采用数据驱动方式，根据数据的可用性来决定指令的执行，而不是由程序计数器来决定执行哪条指令。这种计算机能够充分开发程序中的并行性。

数据驱动计算——指令的执行不受其他控制条件的约束，任何指令只要它所需要的操作数全部齐备且可用时，就可以同时执行。也就是说，指令执行可以相互独立，操作结果可以不受指令执行顺序的影响。

需求驱动计算——只在当某一个函数需要用到某一个自变量时才驱动对该自变量的求值操作。

数据令牌——用来传送数据并激活指令，由结果数值和目标地址等组成。其中，结果数值就是上条指令的运算结果，而目标地址则直接取自上条指令的后继指令地址。

数据流程序图——是一种特殊的有向图，由多个结点以及连接这些结点的有向弧构成。结点中的符号表示进行什么操作。弧代表结点之间的关系及令牌流向。实际上是数据流计算机的机器语言。

2. 问答题

【题 14.2】 答：

(1) 异步性：只要指令所需的数据令牌都已到达，指令即可独立地开始执行，而不必关心其他指令及数据的情况。

(2) 并行性：可同时并行执行多条指令。

(3) 函数性：由于不使用共享的数据存储单元，所以数据流程序不会产生诸如改变存储字这样的副作用，也就是说数据流运算是纯函数性的。

(4) 局部性：运算过程中所产生的数据不是用操作数的地址来引用，而是作为数据令牌直接传送，因此数据流运算没有产生长远影响的结果，其运算具有局部性。

【题 14.3】 答：主要由两部分组成：操作包、数据令牌。

操作包通常由操作码、一个或几个源操作数以及一个或几个后继指令地址组成。后继指令地址用于组成新的数据令牌，以便把本条指令的运算结果送往需要它的目标指令。

数据令牌通常由结果数值和目标地址等组成。其中，结果数值就是上条指令的运算结

果,而目标地址则直接取自上条指令的后继指令地址。

【题14.4】 答:VAL语言具有以下优点。

(1) 并行性好。VAL语言易于开发程序中隐含的和显式的并行性,它提供了相应的语句结构来表达算法中的并行成分,从而能够高效地编写数据流程序。

(2) 遵循单赋值规则。VAL语言没有传统计算机上所用的变量的概念,仅有数值的名称,运算不产生副作用。单赋值使程序清晰,易于理解,为程序的并行执行提供了一种新方法。

(3) 有丰富的数据类型。基本数据类型有:整型、实型、布尔型和字符型等,结构类型有数组和记录等。而且允许数组和记录之间互相嵌套调用,嵌套的深度不限。

(4) 是一种强类型语言。任何函数的自变量和计算结果的数据类型都要在函数的首部加以定义。编译程序在编译过程中能够很方便地检测出函数和表达式中数据类型发生的错误。

(5) 在源程序中不规定语句的执行顺序,没有GOTO之类的程序控制语句。语句的执行顺序不影响最终运算结果。

(6) 编制的程序具有模块化结构。

【题14.5】 答:静态数据流计算机的数据令牌未加标记,当数据令牌在数据流程序图的有向分支线上流动时,在任何一个时钟节拍内,任何一条分支线上只允许传送一个数据令牌。静态数据流计算机的结构比较简单,但对程序并行性的支持不够。

动态数据流计算机的数据令牌带有标记,当数据令牌在数据流程序图的有向分支线上流动时,同一条分支线上可以同时有几个数据令牌在移动。赋予标记的方法有利于最大限度地开发程序中的并行性。如果程序是循环的,则标记方法可以区分出不同层次的迭代执行,实现它们的并行计算。

【题14.6】 答:使用I结构存储器可以避免过多的数组复制操作,节省大量的存储空间和辅助操作开销。

【题14.7】 答:

(1) 高度的操作并行性。

数据流计算机可以实现操作的高度并行性,它不仅能够开发程序中有规则的并行性,而且可以开发程序运行中隐含的并行性。

(2) 流水线异步操作。

数据流计算机中的指令是直接使用操作数本身,实现的是无副作用的纯函数型程序设计方法。可以在过程级和指令级充分开发程序中的异步并行性,可以把串行计算问题用简单的方法展开成并行计算问题来处理。

(3) 与VLSI技术相适应。

数据流计算机结构的基本组成所具有的模块性和均匀性正好与VLSI技术相适应。其中的指令存储器、数据令牌缓冲器、可执行指令队列缓冲器等存储部件都可以采用VLSI技术制造的存储阵列均匀地构成。处理部件和信息包开关网络也可以分别用模块化的标准单元有规则地连接构成。因此,完全有可能充分利用VLSI技术,研制出具有很高性能价格比的数据流计算机。

(4) 有利于提高程序设计效率。

数据流计算机使用的是基于纯函数操作的程序设计语言,它完全摒弃了传统编程语言

所依赖的变量和变量赋值机制，从而彻底消除了编程中使用全局变量和同名变量所产生的副作用。另外，由于函数程序设计语言有更高的自动向量识别能力，也使得机器对数据流的分析和处理更为有效。

3. 应用题

【题 14.8】

解：数据流程序图的执行过程如图 14.6 所示。

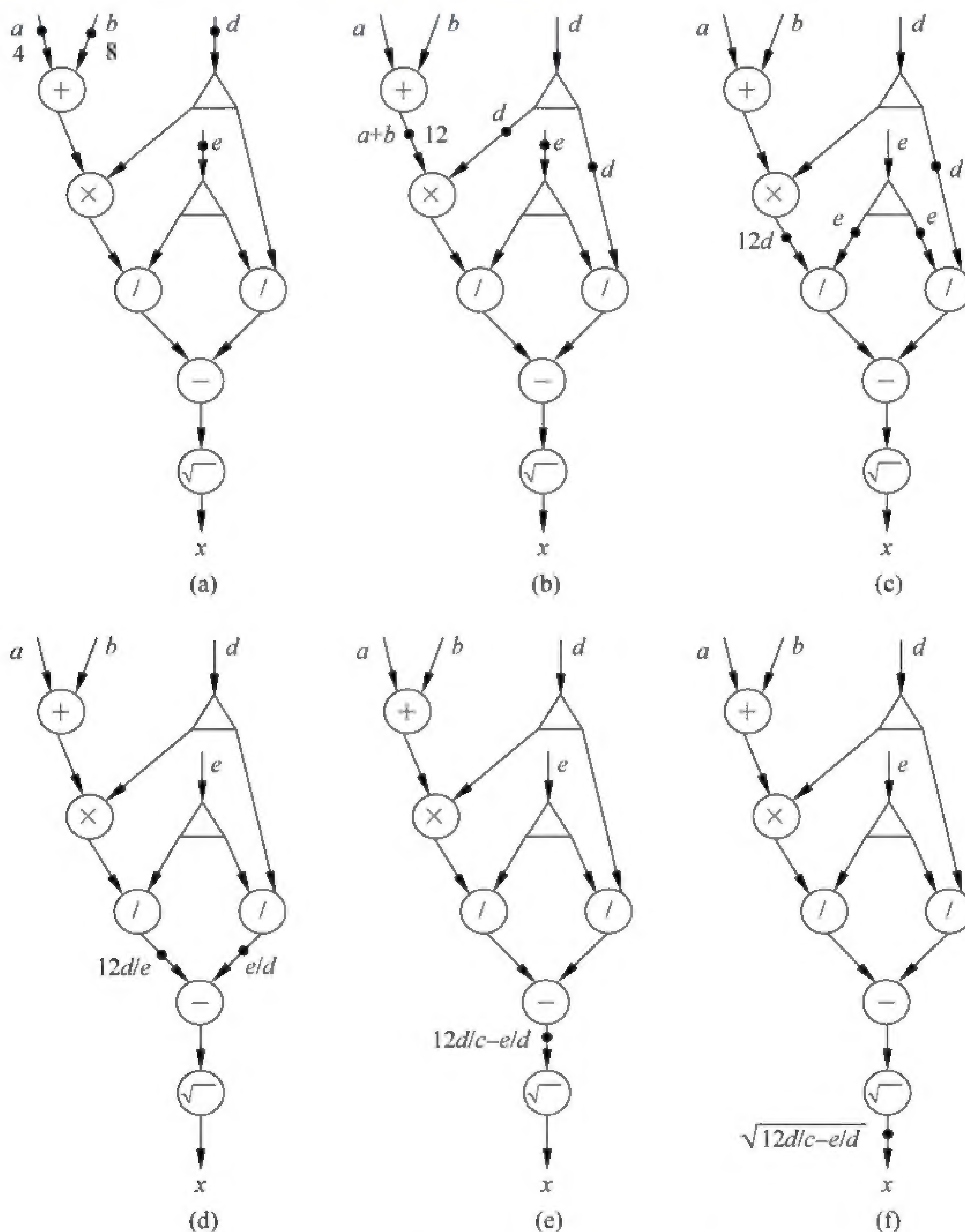


图 14.6　数据流程序图的执行过程

【题 14.9】

解：数据流程序图如图 14.7 所示。

【题 14.10】

解：数据流程序图如图 14.8 所示。

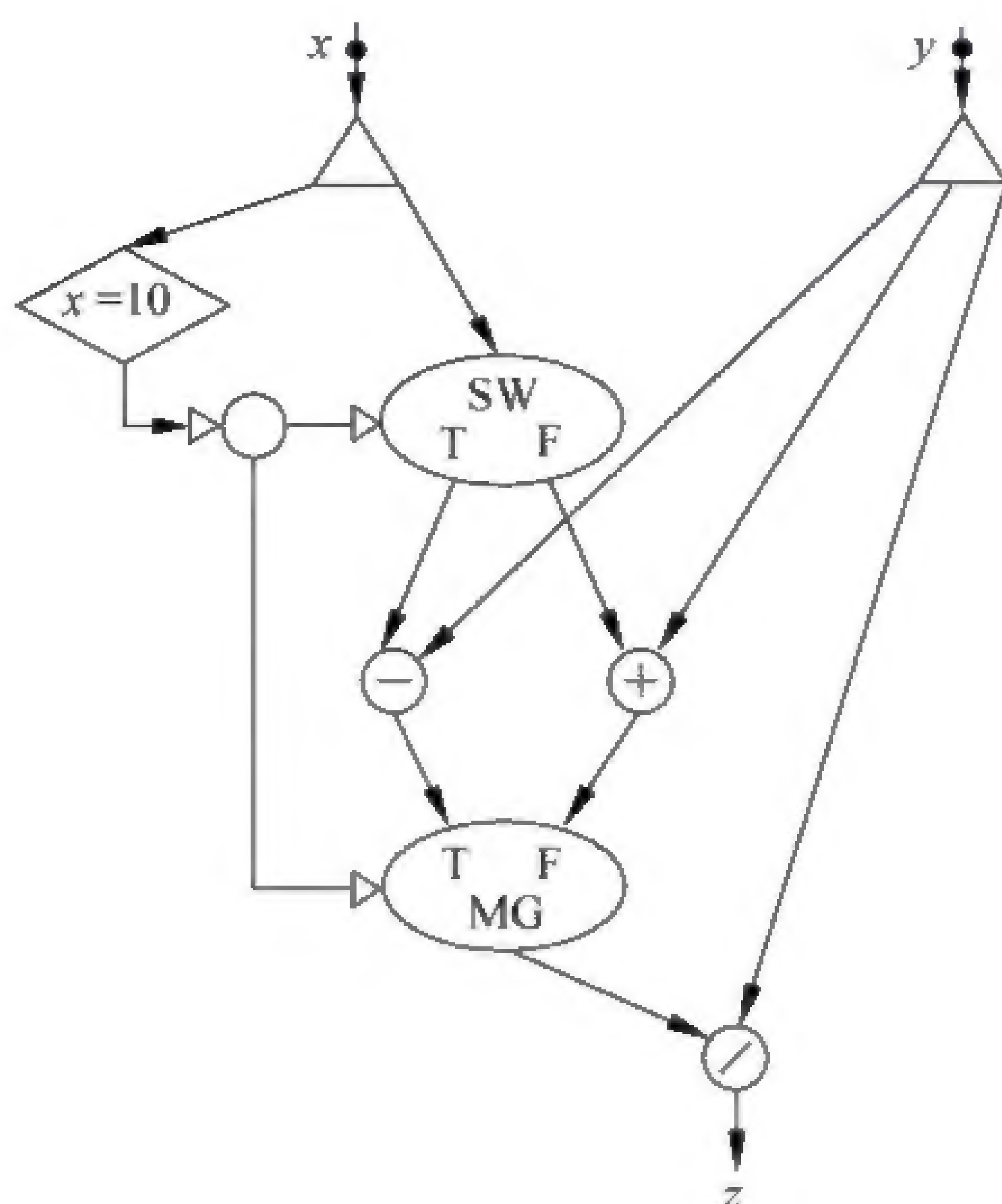


图 14.7 数据流程序图

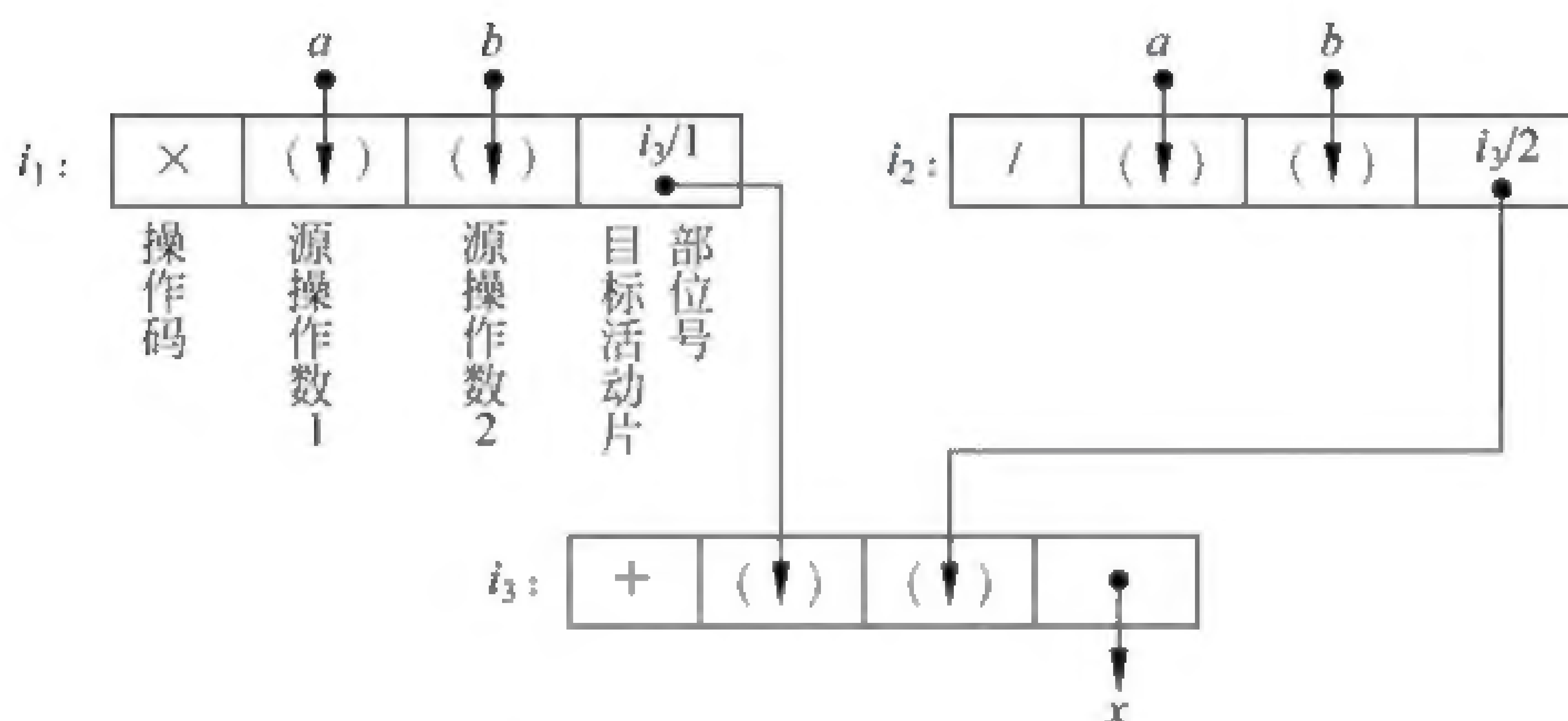


图 14.8 数据流程序图

【题 14.11】

解:如图14.9所示,利用一个复制结点,一个T门控结点和一个F门控结点实现起始数据令牌的两路传送,它根据起始控制令牌所携带的是真值还是假值把起始数据令牌分别送往G1数据流程序图或G2数据流程序图,并利用一个归并门控结点选择G1或G2数据流程序图中的一个结果作为输出,选择的依据仍然是起始控制令牌携带的是真值还是假值。

【题 14.12】

解:如图14.10所示,为了使数据流程序图中的循环体G能够开始执行,在一开始要输入一个起始数据令牌和一个起始控制令牌,并用一个归并门控结点取得循环体G的输入数据令牌。在第一次循环开始时从外部输入数据令牌,而在以后的各次循环中都从循环体本身的输出端取得所需要的输入数据令牌。在第一次循环时,由一个T门控结点控制起始数据令牌是否输入给循环体G。控制方法是起始控制令牌携带真值还是携带假值。另外,用一个判定操作结点根据循环结束条件P产生的控制令牌来控制循环的执行。最后用一个开关门控结点分配每次循环产生的结果数据令牌。如果循环还没有结束,则判定操作结点

P 输出为“真”，通过开关门控结点把数据令牌分配给循环体 G，继续进行下一次循环；如果循环结束，则判定操作结点 P 输出为“假”，通过开关门控结点把结果数据令牌输出到外部。

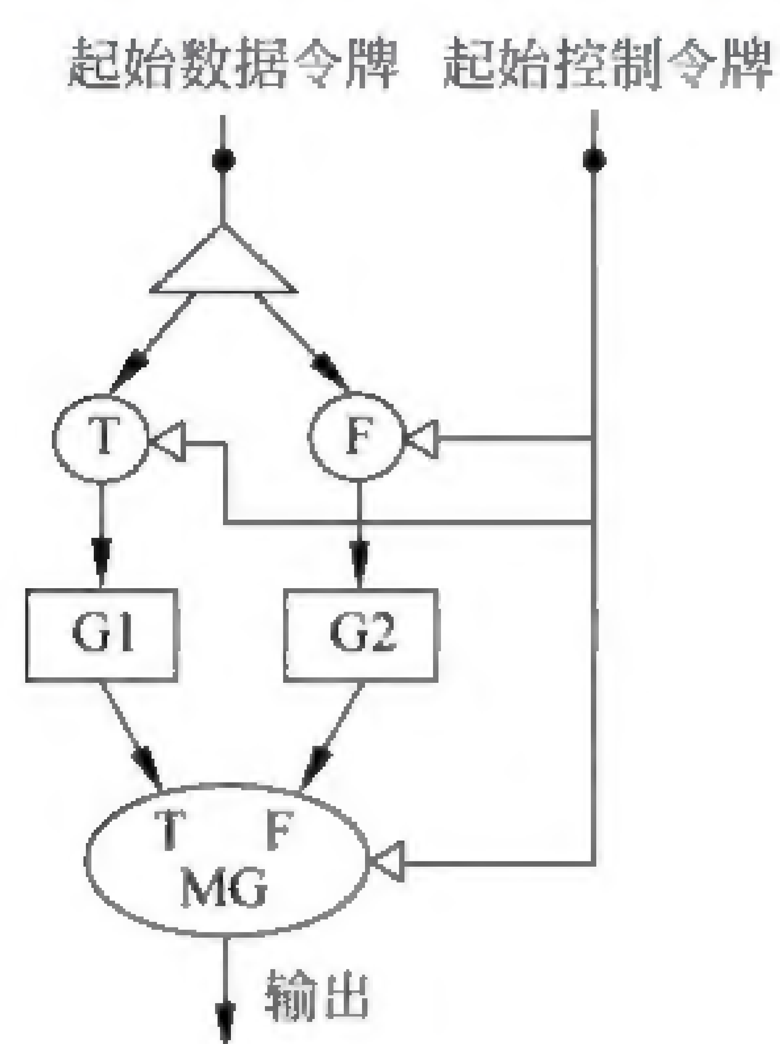


图 14.9　数据流程序图

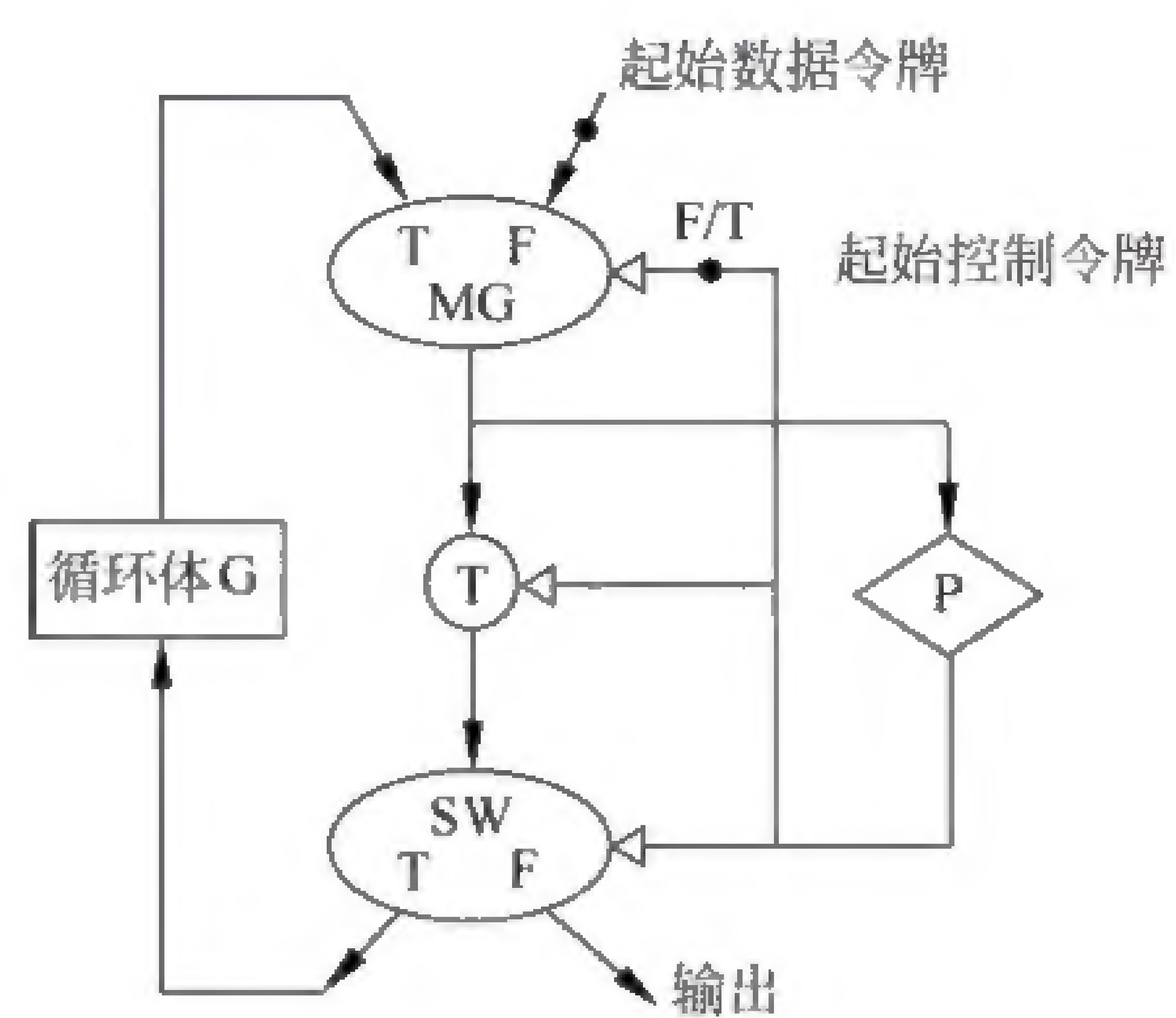


图 14.10　数据流程序图

参 考 文 献

[1] [美]亨尼西(Hennessy J L). 计算机体系结构:量化研究方法(英文版·第5版)(Computer Architecture:A Quantitative Approach,Fifth Edition). 北京:机械工业出版社,2012.

[2] David Culler, et al.. Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann, 1998.

[3] [美]David A Patterson. 计算机组成与设计(硬件/软件接口)(MIPS版,英文版,第5版)(Computer Organization and Design:The Hardware/Software Interface). 北京:机械工业出版社,2014.

[4] 张晨曦等. 计算机系统结构(第2版). 北京:高等教育出版社,2014.

[5] 张晨曦,王志英等. 计算机系统结构教程(第2版). 北京:清华大学出版社,2014.

[6] 郑纬民,汤志忠,计算机系统结构(第2版). 北京:清华大学出版社,2004.

[7] 陈国良等. 并行计算机体系结构. 北京:高等教育出版社,2002.

[8] 李学干. 计算机系统结构(第五版). 西安:西安电子科技大学出版社,2011.

[9] 白中英. 计算机组成与系统结构(第五版·立体化教材). 北京:科学出版社,2017.

[10] 徐炜民,严允中. 计算机系统结构(第3版). 北京:电子工业出版社,2011.